FPGA-based Architectures for Next Generation Communications Networks

Declan Hegarty BA BAI MIEEE MIET

A thesis submitted to
The Universities of
Edinburgh
Glasgow
Heriot Watt
Strathclyde

for the Degree of
Doctor of Engineering in System Level Integration

Abstract

This Engineering Doctorate concerns the application of Field Programmable Gate Array (FPGA) technology to some of the challenges faced in the design of next generation communications networks. The growth and convergence of such networks has fuelled demand for higher bandwidth systems, and a requirement to support a diverse range of payloads across the network span.

The research which follows focuses on the development of FPGA-based architectures for two important paradigms in contemporary networking - Forward Error Correction and Packet Classification. The work seeks to combine analysis of the underlying algorithms and mathematical techniques which drive these applications, with an informed approach to the design of efficient FPGA-based circuits.

Declaration of originality

I hereby declare that the research recorded in this dissertation and the dissertation itself was composed and originated entirely by myself at the Institute for System Level Integration and Aliathon Ltd.

Parts of this work have previously been published as "*An FPGA-based Configurable Network Interface Card*", in the Proceedings of the IEEE International Conference on Systems (ICONS), April 2006.

_____

Declan Hegarty, 17th October 2007.

Acknowledgements

I would like to express my sincere thanks to the principal supervisors of this project, Prof. Joe Sventek at the University of Glasgow and Steve McDonald at Aliathon Ltd., for their assured direction, enthusiasm and for many lively discussions over the last four years, which have inspired and improved this work.

Thanks also to Sandie Buchanan, Siân Williams, Jed Martens, Dr. Wim Vanderbauwhede, Prof. Ivan Andonovic, Prof. Istvan Gyongy, Dr. Chris Athorne, Dr. Noel Smyth, Dr. George Burns and Graeme Kelly.

I would like to thank my wife Nancy for her patience, understanding and encouragement whilst this work was completed.

Finally, to my family - Mum, Fiona, Gavin and all the O'Kanes, thank you.

Contents

Declaration of originality . . . iii
Acknowledgements . . . iv
Contents . . . v
List of figures . . . ix
List of tables . . . xii
Acronyms and abbreviations . . . xiii

1 Introduction **1**
1.1 System Level Integration . . . 1
1.2 ASICs, ASSPs and FPGAs . . . 3
1.3 Thesis Statement . . . 5
1.3.1 On Forward Error Correction . . . 6
1.3.2 On Packet Classification . . . 7
1.4 Dissertation Structure . . . 8

2 Background **9**
2.1 Introduction . . . 9
2.2 Next Generation Communications Networks . . . 9
2.2.1 From PDH to SONET/SDH and OTN . . . 9
2.2.2 From Circuit-Switched to Packet-Switched Networks . . . 11
2.3 Technical Background . . . 12
2.3.1 An Introduction to FPGAs . . . 12
2.3.2 The FPGA and ASIC/ASSP Design Flows . . . 14
2.3.3 Design Trade-Offs and Guiding Principles . . . 17
2.3.4 An FPGA-based Configurable Network Interface Card . . . 20

3 Forward Error Correction **23**
3.1 Introduction . . . 23
3.2 Finite field theory . . . 24
3.2.1 Groups . . . 24
3.2.2 Fields . . . 24
3.2.3 Polynomials over Galois Fields . . . 26
3.2.4 Construction of Extension Fields Based on $GF(2)$ . . . 27
3.3 Coding Overview . . . 30
3.3.1 Useful Properties of Block Codes . . . 30
3.3.2 From Single to Multiple-Error-Correcting Codes . . . 33
3.3.3 From Binary to Non-binary Codes . . . 34
3.4 Forward Error Correction in the Optical Transport Network . . . 36
3.5 Forward Error Correction for 43Gbps Systems . . . 38
3.5.1 Reed-Solomon Encoding . . . 39
3.5.2 A New Approach to Reed-Solomon Encoding . . . 40
3.5.3 A Modified Reed-Solomon Encoder for OTU-3 . . . 44

3.5.4 Reed-Solomon Decoding . . . 47
3.5.5 Syndrome Calculator . . . 47
3.5.6 Key Equation Solver (KES) . . . 49
3.5.7 Chien Search and Forney Calculation . . . 51
3.5.8 Practical Reed-Solomon Decoding . . . 52
3.5.9 Towards Optimisation of the KES Architecture . . . 53
3.5.10 The Hybrid Decoding Algorithm . . . 54
3.5.11 VLSI Implementation . . . 58
3.5.12 Towards an improved KES architecture . . . 59

4 Packet Classification 67
4.1 Introduction . . . 67
4.1.1 String Matching . . . 68
4.1.2 Longest Prefix Matching . . . 68
4.1.3 Exact Matching . . . 70
4.2 Complexity in Packet Classification . . . 70
4.2.1 Space, Time and Power Complexity . . . 71
4.2.2 Update Complexity . . . 71
4.2.3 Massive Linearity and Massive Parallelism . . . 71
4.2.4 Towards the Middle Ground . . . 72
4.3 Techniques for String Matching . . . 73
4.3.1 Aho-Corasick String Matching . . . 73
4.3.2 Boyer-Moore String Matching . . . 75
4.3.3 Contemporary String Matching Solutions . . . 75
4.4 Techniques for Longest Prefix Matching . . . 77
4.4.1 Recursive Flow Classification . . . 77
4.4.2 Grid of Tries . . . 80
4.4.3 Expanded Tries - From Controlled Prefix Expansions to Tree-Bitmap . . . 83
4.5 Techniques for Exact Matching . . . 87
4.5.1 Trees . . . 87
4.5.2 Hashing . . . 88
4.5.3 Contemporary Hashing Techniques . . . 92
4.6 Towards an FPGA-based Packet Classification Engine . . . 96
4.6.1 Understanding d-left: A Numerical Analysis . . . 97
4.6.2 Existing Numerical Results . . . 100
4.6.3 Numerical Results at Improved Load Factors . . . 101
4.6.4 From Static to Dynamic Systems . . . 103
4.6.5 Overflow Sufficiency . . . 104
4.6.6 A New Study of Dynamic Systems . . . 105
4.6.7 Numerical Results for Dynamic Systems . . . 106
4.6.8 Software Simulation of Dynamic 2-Left Systems . . . 107
4.6.9 Interpreting the Software Simulation Results . . . 113
4.6.10 Defining Dynamic Systems Analytically . . . 117
4.6.11 More General System Use-Cases . . . 118
4.6.12 Simulations with Real IPv4 Data . . . 120
4.7 A Prototype Hardware Implementation . . . 125
4.7.1 Promotion and Memory Efficiency . . . 126

4.7.2 Prototype System Dimensions . . . 130
4.7.3 Basic Circuit Operation . . . 131
4.7.4 Implementing Fast Overflow Search . . . 134
4.7.5 New Circuit Operation . . . 139
4.7.6 Resource Utilisation . . . 140
4.7.7 Numerical, Software and Hardware Results Compared . . . 141
4.7.8 Performance Comparison . . . 145

5 Summary **147**
5.1 Chapter 1 . . . 147
5.2 Chapter 2 . . . 147
5.3 Chapter 3 . . . 148
5.3.1 Reed-Solomon Encoding . . . 148
5.3.2 Reed-Solomon Decoding . . . 149
5.4 Chapter 4 . . . 150

6 Conclusions and Further Work **153**
6.1 On Forward Error Correction . . . 153
6.2 On Packet Classification . . . 155

A An Inversionless Berlekamp-Massey Algorithm **158**

B An Extended Euclidean Algorithm **160**

C An Outline Business Plan **161**
C.1 Executive Summary . . . 162
C.2 Company Overview . . . 163
C.2.1 Introduction . . . 163
C.2.2 Operational Summary . . . 164
C.2.3 Industry Overview . . . 166
C.3 Products . . . 167
C.3.1 The FPGA Advantage . . . 167
C.3.2 Product Portfolio . . . 168
C.4 Competitive Environment . . . 170
C.4.1 ASIC/ASSP Competitor Profiles . . . 170
C.4.2 IP Provider Competition . . . 172
C.4.3 SWOT Analysis . . . 174
C.4.4 Aliathon and the Five Forces Framework . . . 176
C.5 The Market Environment . . . 179
C.5.1 PESTLE Analysis of the Macro-Environment . . . 179
C.5.2 Market Segmentation by Network Space . . . 183
C.5.3 Market Segmentation by Technology . . . 184
C.5.4 Telecom/Datacom Market Fundamental Drivers . . . 185
C.5.5 Market Analysis Highlights and Conclusions . . . 186
C.5.6 Aliathon's Marketing Activities . . . 187
C.6 Strategic Plan . . . 189
C.6.1 Overview . . . 189
C.6.2 Migration to IP Sales . . . 190

C.6.3 Increase Direct Marketing Activity 190
C.6.4 Consolidate Partnerships with Xilinx and Altera 190
C.6.5 Develop Standard Products 190
C.7 Financials 191
C.7.1 Overview 191
C.7.2 Revenue Projection 192
C.8 Supporting Information 193
C.8.1 Background to the Aliathon Portfolio 193
C.8.2 IP Taxonomy 194
C.8.3 Target Customers 196

References **199**

List of figures

1.1 The productivity gap, as defined by the ITRS . . . 2
1.2 FPGA replacement of an ASIC/ASSP ATM-over-T1 solution . . . 5

2.1 Xilinx CLB and slice elements . . . 13
2.2 A simple boolean logic function realised in a LUT . . . 14
2.3 Generalised FPGA (a) and ASIC/ASSP (b) design flows . . . 15
2.4 The problem of combinatorial delay in digital systems . . . 16
2.5 Pipelining the logic of Figure 2.4 to improve timing . . . 18
2.6 A 4-bit comparator coded with low level abstraction . . . 19
2.7 A 4-bit comparator coded with high level abstraction . . . 19
2.8 Top level PCB architecture of an FPGA-based network interface card . . . 21

3.1 Simple modulo-2 operations . . . 25
3.2 Power, polynomial and n-tuple representations of $GF(2^3)$ . . . 29
3.3 A basic communications system . . . 30
3.4 The Hamming $(7, 4)$ block code . . . 31
3.5 A trivial binary code . . . 33
3.6 Non binary code with 8 symbol errors - 1 bit error per symbol . . . 35
3.7 Non binary code 8 symbol errors - 8 bit errors per symbol . . . 35
3.8 OTU*k* frame structure . . . 36
3.9 Byte interleaved FEC as specified by ITU-T G.709 . . . 37
3.10 Circuit for dividing $x^6 + x^5 + x^4 + x^3 + 1$ . . . 39
3.11 Standard $RS(255, 239)$ encoder architecture . . . 40
3.12 OTU-3 encoding based on single symbol encoders . . . 41
3.13 Simple encoder for analysis over two clock cycles . . . 42
3.14 Reformulated two-symbol encoder for $g(x) = x^2 - x + \alpha^{206}$ . . . 44
3.15 Most significant symbols of a Reed-solomon encoder . . . 45
3.16 A modified Reed-Solomon RS(255,239) encoder for OTU-3 . . . 45
3.17 Invalid data overlap in a two-symbol encoder . . . 46
3.18 A typical Reed-Solomon decoder system . . . 47
3.19 Generator matrix for $g(x) = 1 + x + x^3$ . . . 48
3.20 A systematic generator matrix . . . 48
3.21 A simple syndrome calculator circuit . . . 49
3.22 A chien search circuit . . . 52
3.23 Detailed working of hybrid algorithm for the evaluator polynomial . . . 57
3.24 Detailed working of hybrid algorithm for the locator polynomial . . . 57
3.25 VLSI architecture for evaluator computation based on a hybrid decoding algorithm . . . 58
3.26 VLSI architecture for locator computation based on a hybrid decoding algorithm 58
3.27 Non-zero coefficients with 1 symbol error . . . 60
3.28 Non-zero coefficients with 2 symbol errors . . . 60

3.29 Non-zero coefficients with 3 symbol errors . . . 61
3.30 Non-zero coefficients with 4 symbol errors . . . 61
3.31 Non-zero coefficients with 5 symbol errors . . . 62
3.32 Non-zero coefficients with 6 symbol errors . . . 62
3.33 Non-zero coefficients with 7 symbol errors . . . 63
3.34 Non-zero coefficients with 8 symbol errors . . . 63
3.35 Proposed coefficient justification for worst case error vector . . . 64
3.36 Modified KES architecture for a single coefficient . . . 65

4.1 Address grouping with prefix and wildcard . . . 69
4.2 Standard SRAM(a) and TCAM(b) cells . . . 72
4.3 A naïve string matching algorithm . . . 74
4.4 Pre-processed optimised pattern shift . . . 75
4.5 Boyer-Moore pattern shifting . . . 76
4.6 Example lookup using recursive flow classification . . . 79
4.7 Set Pruning (a) and Grid of Tries (b) structure for the classifier of Table 4.2 . . . 81
4.8 Bit Vector structure for the classifier of Table 4.2 . . . 82
4.9 Simple prefix database . . . 83
4.10 A multi-bit trie with Controlled Prefix Expansion . . . 84
4.11 A multi-bit trie with Controlled Prefix Expansion and Leaf Pushing . . . 85
4.12 Node compression by bit vector in the Lulea scheme . . . 86
4.13 Multi-stage lookup as specified by Bennett . . . 87
4.14 Multi-stage lookup implemented in RAM . . . 89
4.15 Memory collision in single hashing . . . 90
4.16 A Bloom Filter . . . 93
4.17 Packet classification with a Counting Bloom Filter . . . 94
4.18 Single hashing (a) and two-random hashing (b) . . . 95
4.19 Expected behaviour of the $x_i(t)$ terms from the 2-left differential equations . . . 100
4.20 Fraction of hash bins with load exactly 2 for a load factor of $\frac{1}{4}$, with deletions/reinsertions at the system capacity M . . . 107
4.21 Fraction of bins with load 2 for varying load factor, with continuous deletion and reinsertion at the system capacity M . . . 108
4.22 Embedded memory utilisation for a 2-left classifier with 16384 items and variable bins, under dynamic deletion and insertion of items . . . 109
4.23 Normalised variance from numerically predicted load for the systems shown in Figure 4.22 . . . 110
4.24 Embedded memory utilisation for a 2-left classifier with 262144 bins and variable items, under dynamic deletion and insertion of items . . . 111
4.25 Normalised variance from numerically predicted load for the systems shown in Figure 4.24 . . . 112
4.26 Embedded memory utilisation for a single hashing classifier with 32768 items and 262144 bins, under dynamic deletion and insertion . . . 115
4.27 Embedded memory utilisation for a *2-left* classifier with 16384 items and 65536 bins, showing bulk deletion and insertion of items . . . 122
4.28 Embedded memory utilisation for a *2-left* classifier with 16384 items and 65536 bins, under more general use-cases . . . 123

4.29 Embedded memory utilisation for a *2-left* classifier with 16384 items and 65536 bins, with IPv4 trace data as input 124
4.30 Simple hash tables - implementation independent (a), and with physical separation between primary hash space and collision resolution overflow (b) . . 127
4.31 Embedded memory utilisation for a *2-left* classifier with 16384 items and variable bins, under dynamic deletion and insertion of items without promotion 129
4.32 Initial prototype classification architecture 132
4.33 Block RAM architecture to facilitate an input comparison every 4 system clock cycles 135
4.34 Parallel comparison of a 4-bit number with 4 others 136
4.35 Cache structures to preserve context accuracy 138
4.36 Revised prototype architecture with 2-level, 2-left hashing 140
4.37 Maintaining correct load counts in a 2-layer, 2-left allocation 141
4.38 Characterising internal loading in a 2-level d-left implementation 142
4.39 Comparison of software simulation and hardware implementation results for a 2-level, d-left classifier with 4096 input items 144

6.1 KES architecture for a single coefficient with improved timing 154
6.2 Fraction of bins with load 2 for varying load factor, with continuous deletion and reinsertion at the system capacity M, for a 3-left (a) and 4-left (b) allocation 156

C.1 Aliathon revenue growth 2001-2006 162
C.2 Total global IP revenue, 2004 - $1.27 Billion 163
C.3 Market for communications chip solutions 2000-2010 164
C.4 Aliathon's position in the communications/networking value chain 166
C.5 FPGA versus ASIC/ASSP in communications systems 167
C.6 Aliathon portfolio and roadmap for telecoms IP 168
C.7 Aliathon portfolio and roadmap for datacoms IP 168
C.8 Wireline ASIC/ASSP Market, 2004 - $6.5 Billion 171
C.9 Porter's forces of competition 177
C.10 Aliathon cores across the network span 185
C.11 Optical Transport Systems Growth 185
C.12 Aliathon revenue trends: IP core sales vs design services 191
C.13 Target Customer Matrix - I 197
C.14 Target Customer Matrix - II 198

List of tables

2.1 SONET/SDH Transmission Rates 10

3.1 OTU type and capacity 38

4.1 A simple classifier dataset 78
4.2 A simple classifier dataset specified over source and destination address only . 80
4.3 Expected fraction of bins with load exactly i with variable number of items allocated into n bins by single hashing (a) and 2-left hashing (b) 101
4.4 Expected fraction of bins with load exactly i with variable number of items allocated into n bins by single hashing (a) and 2-left hashing (b) at improved load factors 102
4.5 Embedded memory requirements of *2-left* and single hashing classifiers for simulated systems at varying load factors 116

6.1 Resource utilisation for OTU-3 FEC architectures 155

C.1 Aliathon customer matrix 169
C.2 Market size and forecasts in key technology segments 187
C.3 Aliathon IP core pricing 2005 188
C.4 Aliathon's projected revenues by selected quarter 193

Acronyms and abbreviations

FPGA	Field Programmable Gate Array
3G	Third Generation (Mobile Telephony)
ASIC	Application Specific Integrated Circuit
ASSP	Application Specific Standard Product
ATCA	Advanced Telecommunications Computing Architecture
ATM	Asynchronous Transfer Mode
CIDR	Classless Inter Domain Routing
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
DDR	Double Data Rate
FEC	Forward Error Correction
FPGA	Field Programmable Gate Array
GF	Galois Field
GFP	Generic Framing Protocol
IC	Integrated Circuit
IP	Intellectual Property or Internet Protocol
ITRS	International Technology Roadmap for Semiconductors
ITU	International Telecommunication Union
KES	Key Equation Solver
LFSR	Linear Feedback Shift Register
LCAS	Link Capacity Adjustment Scheme
MAC	Media Access Control
MPLS	Multi-Protocol Label Switching
MSPP	Multi-Service Provisioning Platform
NEM	Network Equipment Manufacturer
NIDS	Network Intrusion Detection System

OTN	Optical Transport Network
OTU	Optical Transport Unit
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PDH	Plesiochronous Digital Hierarchy
POS	Packet-over-SONET
PWE3	Pseudo-Wire Emulation End-to-End
QDR	Quad Data Rate
QoS	Quality of Service
RAM	Random Access Memory
RFC	Recursive Flow Classification
RTL	Register Transfer Level
SDH	Synchronous Digital Hierarchy
SLI	System Level Integration
SoC	System on Chip
SONET	Synchronous Optical Network
TCAM	Ternary Content Addressable Memory
TDM	Time Division Multiplexed
VCAT	Virtual Concatenation
VHDL	(Very High Speed Integrated Circuit) Hardware Description Language
VLSI	Very Large Scale Integration
VoIP	Voice-over-Internet Protocol
VPN	Virtual Private Network
WDM	Wavelength Division Multiplexed

Chapter 1
Introduction

The semiconductor industry is arguably one of the most dynamic and demanding on our planet. There is an unrelenting emphasis placed on being the first to reach the next perceived technological milestone, and billions of dollars of research and development budget are expended to secure the prestige, and ultimately market share, which this technical leadership position affords. Such massive levels of investment have allowed the industry to prosper, and to go on producing increasingly complex products within ever shrinking design cycle times. Perhaps the most succinct summary of this increased complexity has been accredited to Gordon Moore [1, 2], who correctly predicted that from 1975 the number of transistors which could be integrated per unit area would double approximately every two years.

1.1 System Level Integration

This exponential increase in semiconductor density has ultimately given rise to the discipline of System Level Integration (SLI), acknowledging that contemporary integrated circuits are no longer merely components in Printed Circuit Board (PCB) architectures, but complex microsystems in their own right. The International Technology Roadmap for Semiconductors (ITRS) [3] has identified an emerging *productivity gap* as shown in Figure 1.1 - illustrating the discrepancy between transistor density (shown by the solid line) and the number of transistors which may be incorporated into a design in a staff month (shown by the dashed line). System Level Integration is an attempt to bridge this gap, and is in fact a very broad generalisation incorporating a wide variety of disciplines, all in some respect evolved to handle the complexities of systems with millions of transistors. These include, at varying levels of abstraction:

- *Device Modelling and Design*: Complementary Metal Oxide Semiconductor (CMOS) transistor devices are thought to represent 75% of the world's current semiconductor consumption [3]. Progress in computing technology is therefore heavily dependent on

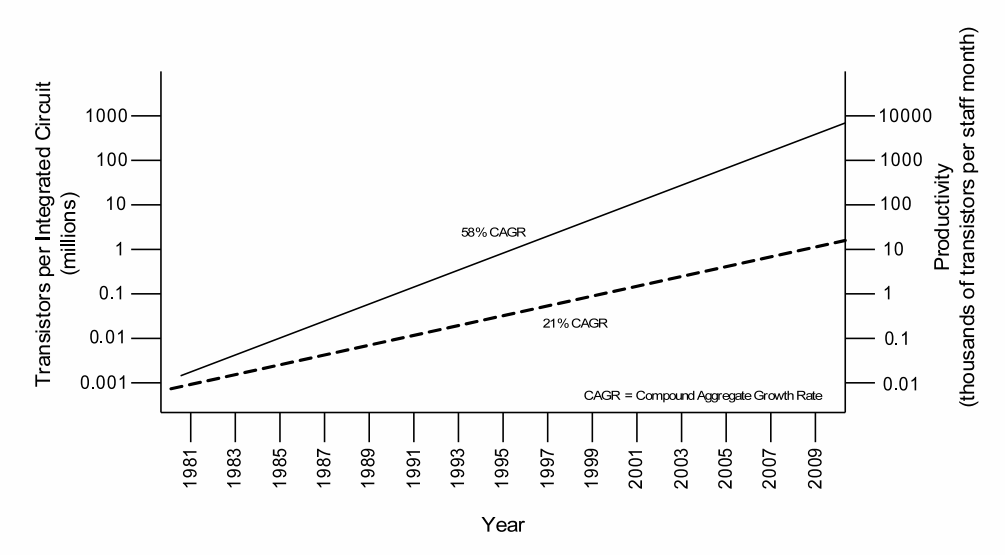

Figure 1.1: *The productivity gap, as defined by the ITRS*

progress in the development of smaller, faster and less power hungry devices. Further, the ITRS predicts that most of the currently known technological capabilities of planar CMOS will approach or have reached their limits by the end of 2016, such that ongoing research in this area is essential for the industry to survive.

- *Block-Based Design and IP-Reuse*: The difficulties of complex systems design are compounded by the competitive markets into which these systems are sold. Time-to-market pressures have led the industry to place an increased emphasis on design re-use [4] whereby, rather than customize systems on a design by design basis, one seeks to develop building blocks (often referred to as Intellectual Property (IP) blocks or cores) which may be reused across multiple designs. These blocks may themselves comprise many hundreds of thousands of transistors, but are designed along best-practice guidelines such as the Quality IP (QIP) Metric [5] to ensure portability between systems.

- *Test and Verification*: The complexity of contemporary System-On-Chip (SoC) designs, and the large number of operational use-cases for the equipment in which they are embedded, mean that ad-hoc testing by the system designer is no longer acceptable. Rather, testing and verification is becoming an industry itself, with numerous dedicated verification languages and test coverage tools emerging to ensure that designs with millions of transistors are appropriately stress tested before release.

Verification and design reuse are in fact often combined, in the form of reusable testbench code, or generic verification blocks which may be instantiated across different (SoC) topologies [6].

The list does not end there of course; the block-based approach has been extended in the form of reusable platforms [7], providing sockets for IP blocks and standardised interconnect; software complexity increases commensurate with logic density; thermal issues become increasingly significant as millions of switching transistors dissipate tens of Watts of power, and packaging, power-distribution and signal-integrity design are major disciplines in their own right - System Level Integration is all of these.

1.2 ASICs, ASSPs and FPGAs

In practice, three categories of System-On-Chip device currently dominate the industry. Pioneered in the early 1980s as transistor arrays of a few thousand logic gates, Application Specific Integrated Circuits (ASICs) are devices which are customised for their end applications, and have fixed functionality once manufactured. Typically, they are commissioned or designed by a single end user. Contemporary ASICs are distinguished from commodity integrated circuits by their complexity, with gate counts in the tens or hundreds of millions.

Application Specific Standard Products (ASSPs) are also typically complex fixed-functionality devices, but are sold as standardised components to multiple customers. In contrast, Field Programmable Gate Arrays (FPGAs) are completely reprogrammable devices, comprising standardised logic blocks and interconnect which the end-user may reconfigure to target a variety of end applications.

The strengths and weaknesses of custom approaches (ASIC and ASSP) versus reprogrammable approaches (FPGA) have been debated for some years [8], and are discussed in more detail in the context of their respective design flows in Chapter 2. On one hand ASICs and ASSPs offer the prospect of higher raw clock speed performance, and are cheaper in high volume. ASIC designs in particular can be very highly optimised and integrate specialist analogue functions. FPGAs however, offer more predictable design cycle times, reduced verification effort, lower non-recurring engineering (NRE) costs, and ease of upgrade.

This project is co-sponsored by an FPGA Intellectual Property supplier - Aliathon Limited - and focuses on the application of FPGA technology to emerging design challenges in networking and communications.

New technologies such as wireless internet access (Wi-Fi and WiMAX), Voice-over-IP, virtual private networks (VPNs) and third generation mobile (3G) are generating strong demand for bandwidth and guaranteed quality of service in contemporary networks. As a result, the traditionally disparate disciplines of voice and data communications are converging to deliver what are becoming known as the "triple play" services: voice, video and data. Network elements deployed to provide these services require integrated circuit (IC) solutions of commensurate sophistication, which have been traditionally offered as ASIC and ASSP devices.

The risks and non-recurring engineering costs of developing such ASICs are significant (reported at US$30M for 90nm fabrication [9]) particularly for lower volume, higher complexity networking applications. Customer needs are extremely diverse and continuously evolving as new standards emerge. Thus ASIC vendors must seek to migrate from a single customer model, by accommodating a broad range of functionality to appeal to a wider customer base, and incorporating some degree of reconfigurability to track evolving standards [10]. The end result is often a large, power-hungry device which may be sub-optimal for a given customer's application.

In contrast, the inherently reprogrammable nature of FPGAs significantly mitigates up-front design and verification costs, and facilitates future-proof systems where new designs can be downloaded to accommodate evolving customer requirements. Functionality can be tailored on an application-by-application basis, eliminating redundant hardware complexity. These advantages encapsulate Aliathon's value proposition. The company seeks to target existing ASIC and ASSP solutions in high value, low volume communications applications for replacement with lower cost, lower power FPGA-based solutions.

Figure 1.2 illustrates a current deployment following this business model - the replacement of multiple Application Specific Standard Products (ASSPs) in an ATM-over-T1 application, where Synchronous Digital Hierarchy (SDH), Plesiochronous Digital Hierarchy (PDH) and Asynchronous Transfer Mode (ATM) blocks have been integrated onto a single FPGA[1].

[1] Additional background on these standards is presented in Chapter 2.

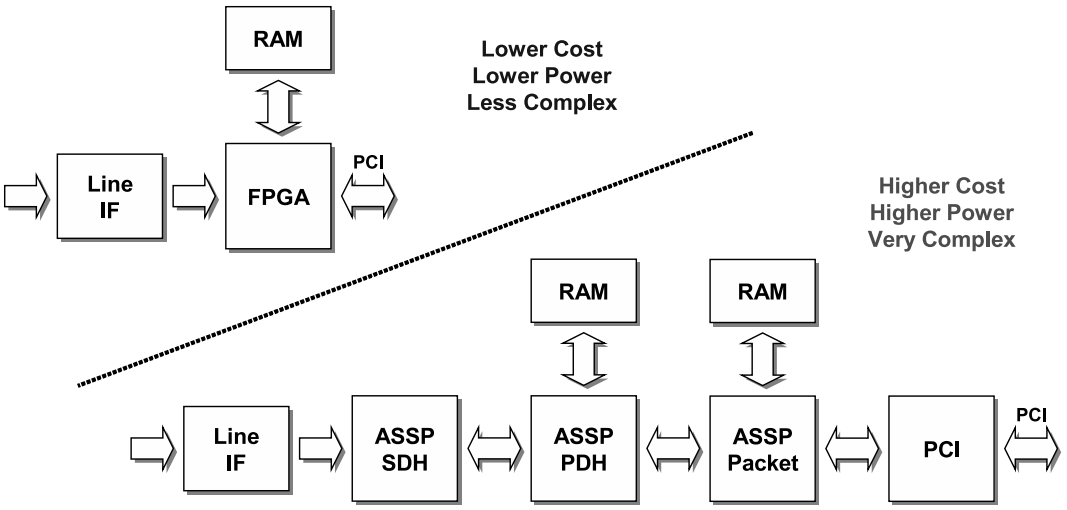

Figure 1.2: *FPGA replacement of an ASIC/ASSP ATM-over-T1 solution*

The viability of this business model is clearly dependent on the company's ability to produce efficient FPGA solutions, which enable chip-count reduction and lower material costs for customers. This in turn is dependent on the efficiency of the component IP cores on which these top-level FPGA solutions are based. Aliathon thus places strong emphasis on technology performance, seeking to produce IP cores which are smaller and faster than those of their competitors.

Creating and maintaining this advantage for next-generation networking and communications applications is the key commercial motivation for the work which follows. The emphasis will be on seeking architectural-level gains, since reusable lower level structures such as arithmetic blocks (multipliers, adders), counters and first-in-first-out buffers (FIFOs) have all been previously developed in Aliathon and have received significant optimisation effort.

1.3 Thesis Statement

Emerging networking standards require data processing operations which are mathematically intensive, and must be completed at high equipment line rates. It is asserted that theoretical research and detailed operational study of the algorithms which underpin these standards, combined with an architectural-level focus on FPGA design, will yield solutions which better the state-of-the-art.

Two important networking techniques, *Forward Error Correction* and *Packet Classification*, are of particular commercial interest to Aliathon and are examined as existence proofs for this assertion.

1.3.1 On Forward Error Correction

Forward Error Correction (FEC) adds additional information in real time to the original information to be transmitted over a channel such that at the receiver, the combination of the original information and the additional information can correct errors in real time.

FEC is increasingly important in very high bandwidth optical networks at transmission speeds up to 43Gbps where it acts to counter signal-to-noise ratio degradation due to fibre loss, chromatic dispersion and other aberrations, and reduces the requirement for expensive optical repeater equipment. Specifically, in the context of the ITU G.709 standard [11] based on Reed-Solomon (255,239) FEC for OTU-3 43Gbps systems, it is asserted that:

- The arithmetic associated with the Reed-Solomon (255,239) encoding can be reformulated, generalised and mapped onto FPGA hardware in the form of a shift-register division circuit capable of processing two input symbols in every system clock cycle. A 43Gbps encoder based on the reformulated arithmetic removes the need for data buffering and pipelined processing required by the single-symbol processing equivalent.
- A Key Equation Solver associated with the Reed-Solomon (255,239) decoding, based on a hybrid Euclidean/Berlekamp-Massey algorithm [12], can be improved by modifying the initial conditions to remove a redundant processing cycle, and by justification of its non-zero coefficients - which approximately halves the logic resource required for polynomial storage. For 43Gbps systems, the modified Key Equation Solver utilises just 56% of the storage resource and 61% of the arithmetic resource required by a direct implementation of the hybrid algorithm, and utilises just 36% of the total resource required by Aliathon's existing solution, based on [13].

1.3.2 On Packet Classification

Packet Classification is a key technology for a wide range of networking and communications application areas, including Internet Protocol (IP) routing and switching, service level differentiation, network security and flow monitoring. Essentially, one seeks to make some decision on the destination of a data packet based on some portion of its contents - typically the packet header. Classless-Inter-Domain-Routing (CIDR) techniques based on longest prefix matching dominate the literature, but string matching and exact pattern matching techniques are becoming increasingly important. The latter is of particular commercial interest to Aliathon, potentially offering performance improvements for legacy products, and enabling provision of IP cores in new areas such as packet filtering, flow monitoring and Pseudo-Wire Emulation End-to-End (PWE3) [14].

Exact match techniques include schemes based on decision-trees, neural networks, Bloom Filters and hashing. State-of-the-art techniques based on trees offer completely deterministic classification or *lookup* times, but scale poorly to systems with long headers. Conversely, contemporary techniques based on hashing scale better but are non-deterministic, such that worst case lookup times can be poor. A hashing-based algorithm called *d-left*, first formalised by Vöcking [15] and later analysed by Mitzenmacher and Broder using differential equations [16], mitigates the effects of non-determinacy and shows promise as the basis of an FPGA-based classification engine. In this context, it is asserted that:

- The analysis presented in [16] can be extended to consider packet classification systems with dynamic deletion and insertion of items.
- This dynamic mode of operation exhibits stable, repeatable steady-state behaviour which places an upper bound on the amount of FPGA block RAM required to support the system.
- This upper bound enables an FPGA-based packet classification implementation which is effectively deterministic.
- A novel 2-level, d-left scheme, implemented in an FPGA requires 62.5% of the external memory resource, and just 2% of the embedded internal SRAM required by a system of equivalent capacity based on Counting Bloom Filters.

1.4 Dissertation Structure

The remainder of this dissertation is structured as follows: Chapter 2 presents general commercial and technical background to the project, and describes the development of an FPGA-based network interface card, undertaken as a precursor to the principal research work. Chapters 3 and 4 comprise the main research contributions, on Forward Error Correction and Packet Classification respectively. Chapter 5 offers a summary, and Chapter 6 presents conclusions and some suggestions for further work. A business plan composed in fulfilment of the business and management requirements of the EngD is included in the appendices.

Chapter 2
Background

2.1 Introduction

This chapter sets the context for the research which follows, both in terms of the commercial motivations for the work and the FPGA architectures targeted by the proposed solutions. The opening discussions present some background on what exactly is meant by a “next generation” network - a general term which embodies both a migration to higher bandwidth systems, and an industry-wide trend towards more diverse payloads over a packet-based infrastructure. This discussion is followed by a general introduction to FPGA technology and the design principles followed in the course of the research. Finally, a description of the FPGA-based network interface development undertaken as the first phase of this project is presented.

2.2 Next Generation Communications Networks

Historically, communications networks have been based on voice telephony. The frequency spectrum of the average human voice ranges from approximately 30Hz up to 4kHz. Thus, in accordance with Nyquist’s rule, one must sample a voice signal at approximately 8kHz to achieve faithful reproduction of that signal. Traditional Pulse Code Modulation systems use 8 bits to represent each sample, so one requires 8000 x 8 = 64kbps for each voice channel.

2.2.1 From PDH to SONET/SDH and OTN

The entire time division multiplexed (TDM) hierarchy has been built around this data rate. Early networks were based on the Plesiochronous Digital Hierarchy (PDH), which aggregated multiple voice channels together in trunk links running at nominally, but not precisely the same frequency. As traffic demands on the network grew, the limitations of the PDH network emerged.

SONET	SDH	Optical	Line Rate (Mbps)
STS-1	STM-0	OC-1	51.48
STS-3	STM-1	OC-3	155.52
STS-12	STM-4	OC-12	622.08
STS-48	STM-16	OC-48	2,488.32
STS-192	STM-64	OC-192	9,953.28
STS-768	STM-256	OC-768	39,813.12

STS: Synchronous Transport Signal
STM: Synchronous Transport Module
OC: Optical Carrier

Table 2.1: *SONET/SDH Transmission Rates*

Lack of standardisation made interoperability between equipment vendors difficult, network management functionality was limited, and individual voice channels had to be added and dropped via a complex and expensive multiplexing and demultiplexing hierarchy.

The core standards for the Synchronous Optical Network (SONET) and the Synchronous Digital Hierarchy (SDH) [17] were introduced to address these shortfalls, adding rich management overhead and unified network timing, for interconnection over a high speed optical infrastructure, running at the frequencies shown in Table 2.1.

The SONET/SDH infrastructure is now well established, with an installed base of some 390,000 rings worldwide. However, the communications industry is entering yet another period of transition. Increases in voice traffic have been incremental in the last 5 years. Data traffic, by comparison, has increased exponentially, driven by the global adoption of broadband services.

Network providers are thus increasingly seeking more cost-effective ways to provision and manage large amounts of bandwidth and diverse payloads. *Next-Gen* techniques such as Virtual Concatenation (VCAT) and the Link Capacity Adjustment Scheme (LCAS) have emerged [18] to improve SONET/SDH bandwidth utilisation when payload bit-rates do not fit conveniently into the traditional hierarchical containers. The ITU-T G.709 Optical Transport Network (OTN) [11, 19] has also emerged as an improved physical layer protocol, with a streamlined subset of the SONET/SDH management functions, support for the management of Wavelength Division Multiplexed (WDM) systems, and tandem connection monitoring for signal diagnostics across multiple networks. Aliathon currently offers a portfolio of IP cores

targeting OTN applications at 2.7Gbps, with a roadmap to 10.7Gbps and 43Gbps systems - the latter being a key commercial motivation for the research on Forward Error Correction presented in Chapter 3.

2.2.2 From Circuit-Switched to Packet-Switched Networks

In addition to this generalised increase in demand for bandwidth, market reports [20–22] clearly indicate a migration away from legacy TDM networks towards a packet-switched infrastructure, based on the Internet Protocol (IP), which enables network operators to realise a lower cost per bit in the delivery of voice, video and data services. IP is ubiquitous, but is not optimised for transfer of delay sensitive signals such as video and voice. Techniques such as Multi-Protocol-Label-Switching (MPLS) and Diffserv have emerged to address these issues, providing Quality-of-Service (QoS) guarantees over packet-based networks and enabling unified transmission of voice, video and data over the same infrastructure. A 2005 survey of Service Providers in North America and Europe [23] conducted by Infonetics Research indicated the following:

- That service providers faced the double bind of keeping pace with significant growth in IP traffic whilst simultaneously trying to increase the profitability of IP-based services. 77% of respondents identified maintaining revenue growth in a period of technical transition as the key business challenge faced.
- That broadband internet adoption was the key driver for the growth in IP traffic.
- That over 65% of respondents were no longer evaluating legacy Asynchronous Transfer Mode (ATM) or Frame Relay equipment.
- That over 25% of respondents had already deployed IPv6 networks in 2004, although deployments were not on a large scale. By close of 2005, half of all respondents expected to have some IPv6 roll-out.
- That 81% of respondents used some form of MPLS networking in 2004, with 92% adopting the technology by the end of 2005.

So, whilst the installed SONET/SDH infrastructure continues to generate revenue for Aliathon, continuing migration to packet-switched networks by service providers and network equipment

manufacturers underlines the need to develop technology in this area - the commercial motivation for the research on Packet Classification presented in Chapter 4.

2.3 Technical Background

2.3.1 An Introduction to FPGAs

In introducing FPGA technology, the *Virtex-II Pro* Platform FPGA [24] from Xilinx Inc. will be used as a reference, since this device was in full production at the outset of this project, and was thus used both in the prototype platform discussed in 2.3.4, and in analysis of FEC and Packet Classification architectures discussed in later chapters. Newer device families have of course continued to emerge during the course of this research[1], but the architectural fundamentals remain broadly the same.

In the most general context an FPGA consists of an *array* of programmable logic elements arranged on a matrix of programmable interconnect. In the Virtex-II Pro device these elements include high-speed serial transceivers with a bit-rate of up to 3.125Gbps per channel, embedded IBM PowerPC Reduced Instruction Set Computer (RISC) processor cores, embedded multiplier blocks optimised for Digital Signal Processing (DSP), Digital Clock Manager (DCM) blocks for clock distribution and embedded dual-port SRAM.

The fundamental programmable element across all Xilinx device families is the Configurable Logic Block (CLB). Each CLB in turn comprises 4 *slices* as shown in Figure 2.1.

[1] Notably Virtex-4 and Virtex-5 from Xilinx Inc., and Stratix-III from Altera Corp.

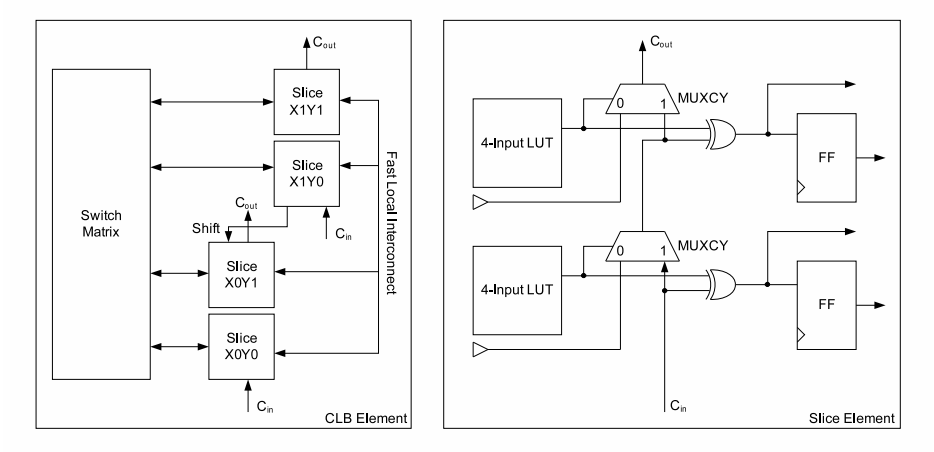

Figure 2.1: *Xilinx CLB and slice elements*

These slices provide the essential structures required for synchronous and combinatorial logic design. Each slice contains two flip-flops - the basic storage elements for synchronous logic designs, and two 4-input Look-Up Tables (LUTs) each capable of implementing any boolean function of four inputs or less, as illustrated in Figure 2.2. The simple 2-input "OR" and "AND" primitives in this example are emulated in the FPGA by using the inputs directly as the LUT address. Each of the 16 LUT locations stores a value required to correctly resolve the logic function for a given input address, such that the LUT is effectively a direct hardware implementation of the truth table associated with the logic function. Understanding this basic 4-input granularity (and indeed the FPGA target architecture in general) is often important in realising efficient FPGA implementations. Perhaps unsurprisingly, authors have noted that Very Large Scale Integration (VLSI) designs optimized for traditional ASIC platforms do not always perform well in FPGA logic and vice-versa [25, 26].

The basic programmable logic elements are supported by a hierarchy of programmable interconnect between *switch matrices*. In the Virtex-II Pro local fast interconnect lines run internally in the CLB connecting LUT outputs to LUT inputs as shown in Figure 2.1. Conceptually above this layer, there are 16 direct connections between neighbouring CLBs, and a series of *double lines* which route signals to every first or second block away in four directions. In similar fashion, *hex lines* route signals to every third or sixth block away in four directions, and finally a series of *long lines* distribute signals across the whole device. Additional dedicated routing is provided for system clocking and high speed arithmetic operations.

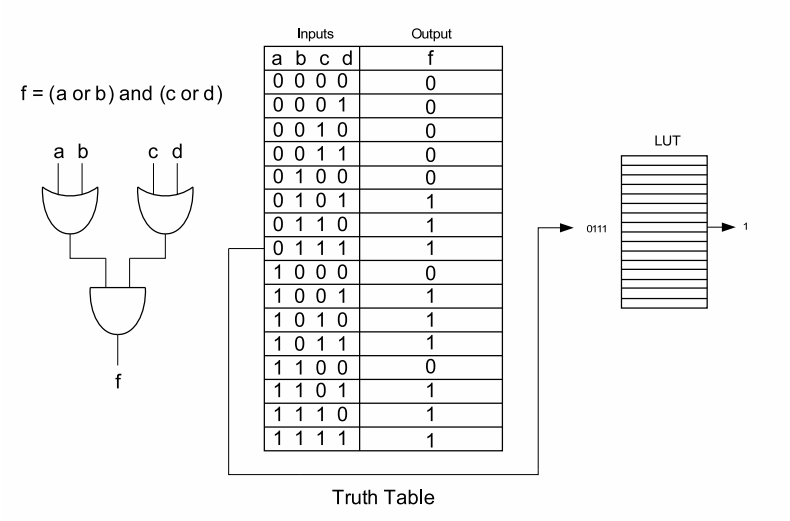

Figure 2.2: *A simple boolean logic function realised in a LUT*

2.3.2 The FPGA and ASIC/ASSP Design Flows

As discussed briefly in Chapter 1, the advantages of an FPGA-based approach over ASIC/ASSP equivalents are perhaps best discussed in the context of their respective design flows, shown in Figure 2.3. Both flows start with a functional specification, and design entry - typically using a Hardware Description Language (HDL). Although schematic-based design entry is still supported by contemporary FPGA design flows, it becomes difficult to manage for all but the simplest designs. At present, Verilog and VHDL ((Very High Speed Integrated Circuit) Hardware Description Language) are the dominant HDLs, the latter used throughout the remainder of this dissertation. A discussion of the relative merits of these two languages is beyond the current scope, although some discussion on appropriate coding style follows in 2.3.3.

During the design capture phase, two main categories of HDL code are typically written in parallel. *Register Transfer Level* or simply *RTL* code encapsulates the design proper, and is intended to be processed in the next stage of the flow by a *synthesis* tool, which maps the functional design onto logic primitives in the target architecture. On the other hand *behavioural* code is not itself intended for synthesis, but rather used in conjunction with a *simulation* tool for functional verification of the design. Such code is usually described as a *testbench*. The early phases of the FPGA and ASIC/ASSP flows thus follow a similar iterative process. The design is specified and coded, verified in a testbench and synthesised.

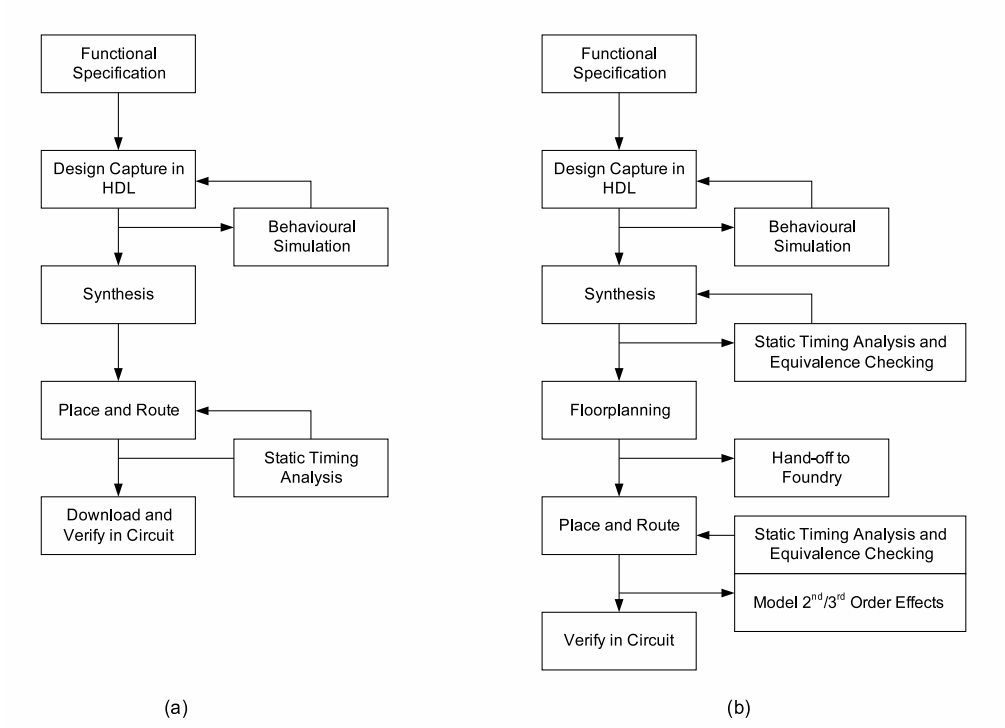

Figure 2.3: *Generalised FPGA (a) and ASIC/ASSP (b) design flows*

Thereafter the flows begin to differ. The high NRE costs associated with ASIC/ASSP fabrication mean that it is extremely important that the design is *exactly* as originally specified before committing to manufacture. Theoretically, after initial functional verification is successfully completed, the design should remain functionally equivalent (and by implication correct) through to manufacture. In reality, the numerous translations and optimisations from RTL to physical design in multi-million gate designs (including insertion of I/O and test structures, logic and drive strength optimisations and manual net list changes) inevitably introduce bugs. As a result, complex formal equivalence checking techniques become necessary in the ASIC/ASSP flow [27, 28] to ensure that functional integrity at every stage in the flow is maintained.

Timing performance is critical in any logic implementation, and achieving timing *closure* a key goal for designers of large digital systems, regardless of platform. The timing challenge is generalised in Figure 2.4. A typical digital system comprises flip-flops (or registers) as the basic storage element, and combinatorial elements (gate primitives in ASICs/ASSPs, LUTs in FPGAs) used to implement the logical and mathematical functions demanded by the functional specification. The storage elements (FF_1 and FF_2 in the figure) hold their current data value

at their outputs until the next clock edge occurs (flip-flops are rising edge triggered in this example) when the outputs are updated with the current input value.

In practice, two things work against the system designer here. Firstly, the storage elements demand that the data on their inputs be held stable for a minimum time before the clock edge arrives (the setup time t_{su}) and for a minimum time after the clock edge arrives (the hold time t_h). Effectively, this means that the data must be sampled within a valid window (shown in grey in the figure). Secondly, combinatorial logic can only operate at finite speed - all technology families suffer a data delay δ, which is typically much larger than any delays on associated clock nets. As shown in Figure 2.4, if this delay becomes too large, timing violations can occur, whereby the clock edge does not fall within the valid sampling window.

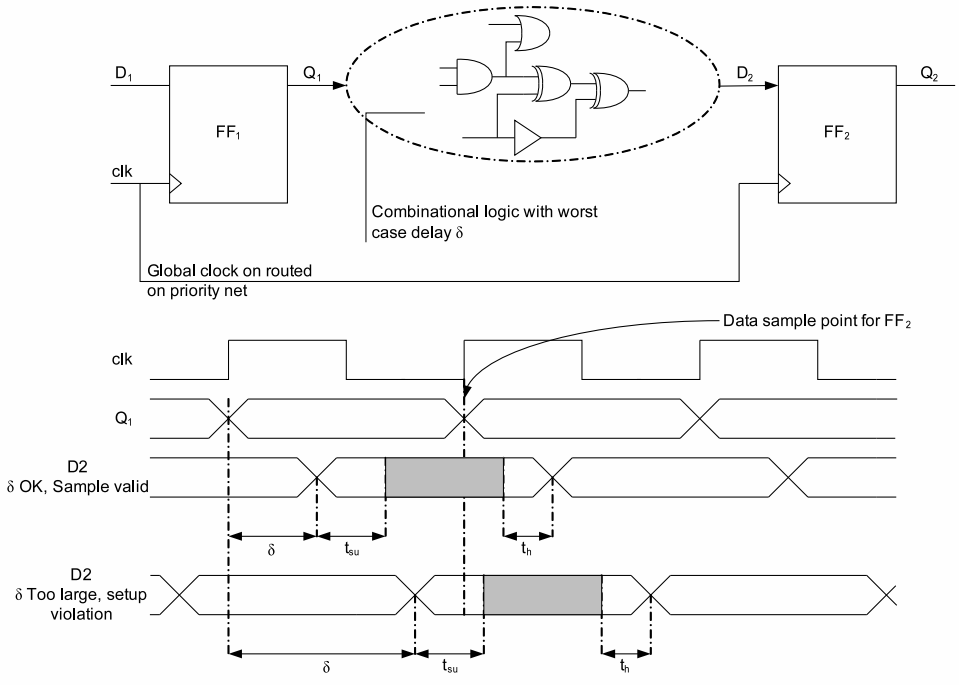

Figure 2.4: *The problem of combinatorial delay in digital systems*

Managing timing issues as system complexity and speed increase is challenging, and *Static Timing Analysis* tools are used in both FPGA and ASIC/ASSP flows to automate the process. The process is simplified in the FPGA domain by the fact that the physical location of the programmable logic elements in the device is fixed, and the parasitics associated with the programmable interconnect have been pre-characterised, such that a single (though often iterative) timing analysis phase suffices. In contrast, the ASIC/ASSP

flow includes a *floorplanning* stage (when the designer decides what logic blocks go where), into which iterative timing analysis must also be folded. Finally, the ASIC/ASSP flow is further complicated by its dependence on foundry availability for device manufacture, which introduces risk and delay (incorporating additional equivalence checking after place and route, and modelling of parasitic effects) before the design can finally be verified in circuit.

The FPGA design flow reflects the fact that the devices are inherently reprogrammable. Design errors can be rectified by downloading a new bitfile, and do not require a silicon re-spin. The FPGA design flow is thus much less verification driven than the ASIC/ASSP equivalent, and simpler as a result. One must be careful not to oversimplify the case here; market volumes for ASICs/ASSPs clearly indicate that they represent the most cost effective technology in many applications. Nonetheless for lower volume, high-complexity networking solutions, where new standards continuously emerge and evolve, FPGAs appear to offer a more cost effective, lower risk solution for many Network Equipment Manufacturers (NEMs) - a fact reflected in Aliathon's current customer base and product portfolio.

2.3.3 Design Trade-Offs and Guiding Principles

Whilst the research projects on Forward Error Correction and Packet Classification which follow in Chapters 3 and 4 by definition target different application domains, they both represent an attempt to optimise FPGA systems by balancing classic design trade-offs. These trade-offs are effectively universal - reflecting the zero-sum game of finite clock speed and available logic resource - and independent of one's target application. Generally speaking, one may trade additional logic resource for system clock speed, though this is subject to diminishing returns as the limits of the process technology are approached.

2.3.3.1 Logic Utilisation, Clock Speed and Processing Time

Consider the timing problems illustrated previously in Figure 2.4. Recall that with a large δ through the combinatorial logic path, the setup time was violated and reliable operation of the flip-flops in this circuit could not be guaranteed. There are two approaches to achieving timing closure for this design; reducing the system clock speed to give a wider sampling window, or reducing the worst case delay through the combinatorial logic until setup and hold times are met. Assuming that the design specification prohibits a slower clock, the worst case

combinatorial delay can be reduced by *pipelining* the logic as shown in Figure 2.5. Pipelining refers to the insertion of additional synchronous elements (flip-flops) into the combinatorial logic path, such that the worst case delays between these elements are reduced.

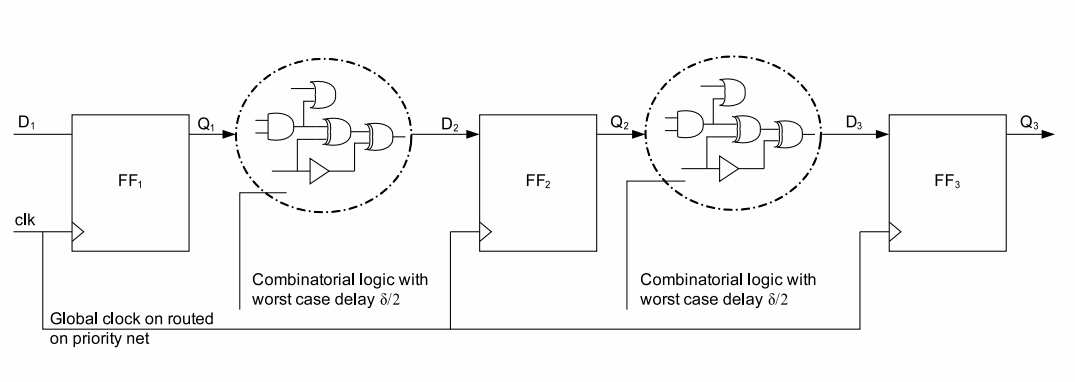

Figure 2.5: *Pipelining the logic of Figure 2.4 to improve timing*

Assuming that the flip-flop is inserted at a point exactly half way along the datapath, the worst case delay in this system is now halved, and the maximum allowable clock frequency effectively doubled. This improvement in clock speed comes at the cost of additional synchronous logic utilisation which is negligible in this example, but could be significant in wide datapaths where multiple bits of additional storage are required. Further, the additional flip-flop in the data path means that it now takes an additional clock cycle for data to pass from the system inputs to its outputs. This increased *latency* can become significant in systems with multiple pipeline stages, particularly if the data is sensitive to absolute delay, as in voice or video applications.

Logic utilisation can also be traded against processing time, in terms of the number of clock cycles required to complete a given computation. For example, say one wanted to multiply two polynomials of length l, a common requirement in encryption or coding systems. One could use a single multiplier and produce one polynomial coefficient result per clock cycle, thus taking l clock cycles to produce the complete result. Alternatively, one could use a bank of l parallel multipliers and produce the same result in a single clock cycle. Clearly the best approach is dependent on the demands of the target specification.

2.3.3.2 Abstraction and Technology Independence

In addition to the trade-offs just discussed, the work which follows is influenced by the following best-practice guidelines followed at Aliathon Ltd. Namely that all VHDL source

code should be written at an appropriate level of *abstraction* and that all VHDL source code should be *technology independent*. To elaborate the first point, consider the very simple 4-bit comparator circuit and VHDL implementation shown in Figure 2.6.

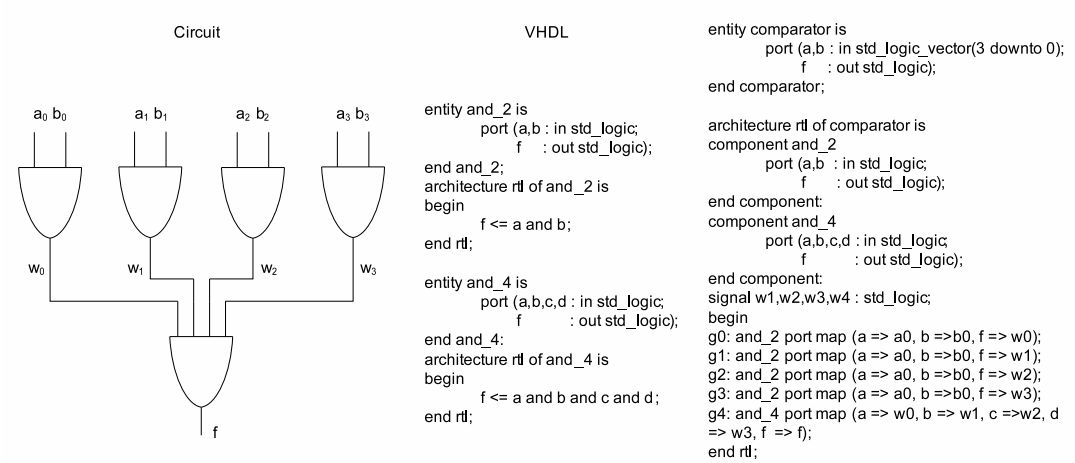

Figure 2.6: *A 4-bit comparator coded with low level abstraction*

The VHDL used here is at a very low level of abstraction - the details of the circuit structure are very closely reflected in the code. Gate primitives are written as components and instantiated in the higher level comparator architecture with interconnecting wires explicitly defined as signals. For all but the simplest circuits this approach quickly becomes impractical and counter-productive, generating large volumes of code. Fortunately, contemporary synthesis tools allow designers to write VHDL at a much higher level of abstraction, with confidence that the hardware structures produced will still be as intended. Thus the comparator code could be re-written as shown in Figure 2.7, with identical results.

```
entity comparator is
        port (a,b : in std_logic_vector(3 downto 0);
              f   : out boolean);
end comparator;
architecture rtl of comparator is
begin
        f <= a = b;
end rtl;
```

Figure 2.7: *A 4-bit comparator coded with high level abstraction*

Not all examples are so clear cut, and what constitutes an appropriate level of abstraction for any given design is subjective. On one hand, coding at too low a level results in large code bases which are difficult to maintain, debug and re-use. On the other, very abstract code

can be difficult to understand, and may not be interpreted as intended by the synthesis tools. Aliathon guidelines thus recommend writing VHDL at the highest level of abstraction which the synthesis tools will support.

Maintaining *technology independence* implies that one must not write HDL which results in a synthesised circuit unique to a particular target architecture. As an example, say one wanted to implement an asynchronous first-in-first-out (FIFO) buffer - an extremely common data structure in digital systems design. The Xilinx Virtex-4 family of devices include integrated FIFO logic, which can be inferred in synthesis by setting appropriate attributes in HDL. Whilst efficient, using this dedicated logic in an IP core immediately locks that core into the Virtex-4 architecture and limits flexibility for Aliathon's end customers. Thus, where possible, all logic structures are implemented using generic resource - available across multiple target architectures - even if this incurs a penalty in terms of absolute logic utilisation.

2.3.4 An FPGA-based Configurable Network Interface Card

As a precursor to the main research phases of this project, an FPGA-based printed circuit board (PCB) network interface prototype was developed [29]. This development was intended principally as a support prototype for the subsequent work on Packet Classification, and is used in Chapter 4 to compare a hardware implementation of a *d-left* based classifier with theoretical predictions and software generated results. Further, at the outset of this project, Aliathon had no in-house platform for hardware verification of their IP cores - a gap which this development was also intended to fill; hence its general purpose infrastructure. The design was taken from concept through schematic capture, PCB layout, manufacture and eventual deployment at beta-test locations in customer installations. The basic system block diagram is shown in Figure 2.8.

The system is built around a Xilinx Virtex II Pro XC2VP50 FPGA, with a supporting infrastructure designed to be applicable to the widest possible range of potential network applications. With the FPGA as its principal component, the PCB architecture is based on a 12 layer stack-up to accommodate high density interconnect - with controlled impedance for good signal integrity on high bandwidth interfaces to the memory components and to the network transmission modules. The system operates from a single 12 volt power supply which is tightly regulated on board using tailored small footprint switch-mode power supplies and additional load regulation for noise-sensitive high speed circuits.

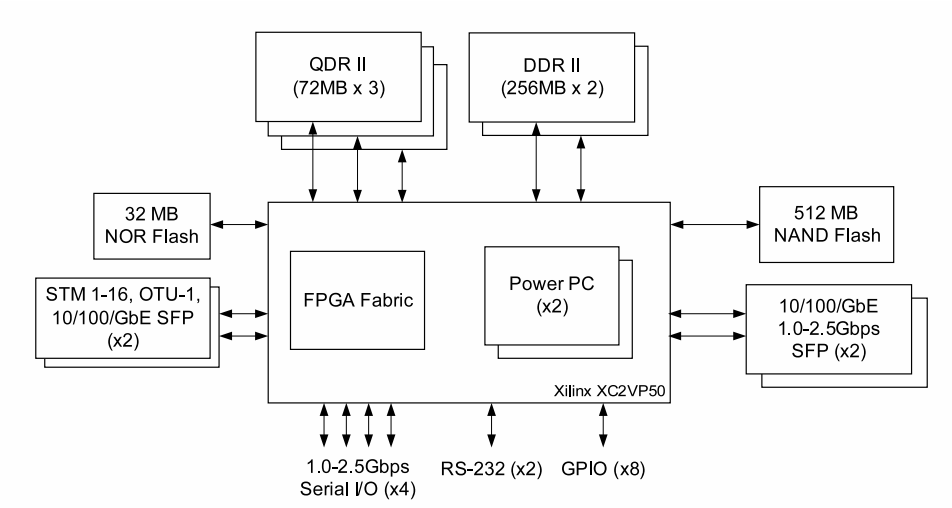

Figure 2.8: *Top level PCB architecture of an FPGA-based network interface card*

Integral to the flexibility and capability of the system is the supporting memory infrastructure. Three 72Mb Quad-Data-Rate II (QDR-II) SRAM devices are directly connected to the FPGA to provide extremely high bandwidth memory access. The data interface to these devices is 32 bits wide, running at 166MHz with access on both edges of the system clock to deliver an aggregate data throughput of 10.624Gbps. Additional memory density is provided by 512MB of Double-Data-Rate II (DDR-II) memory, with access speeds up to 166MHz.

The choice and balance of these emergent memory technologies is fundamental to the range of potential applications the system can support. The high throughput provided by the QDR-II devices facilitates real time, full bandwidth processing of the data on the network interfaces. The DDR-II devices provide superior memory density in an extremely small footprint for buffering large amounts of data, essential in applications such as differential delay measurement or packet storage. Such a balance of speed and density on its memory interfaces ensures that the FPGA can perform close to optimally across a wide range of communications functions. In addition, PCB footprints for the memory architecture have been carefully chosen to allow the memory capacity of the system to be easily expanded as latest devices are released, or reduced for lower cost solutions to less demanding applications.

Two fully configurable optical interfaces are based on Small-Form-Factor Pluggable (SFP) transceiver modules which mate directly with the optical fibre. These modules integrate the transimpedance amplifier and laser diode driver required for optical transmission in

an extremely compact footprint. The system is thus compatible with all the key optical networking standards up to 3.125Gbps. The SFP modules can be easily swapped out of their host connectors, giving the platform a completely general purpose front-end interface, easily configured for a host of optical performance requirements ranging from long range Dense Wavelength Division Multiplexing (DWDM) systems through to low cost, single rate short reach applications.

Two further SFP modules serially connected to the FPGA provide additional high bandwidth access over fibre or copper. These ports are again designed to be completely general purpose; optimal for configuration as 10/100/Gigabit Ethernet (GbE) electrical interfaces for remote redundant operation of the platform, but equally amenable to optical connectivity. A fully configurable backplane interface is provided by four programmable 3.125Gbps channels, output via a high frequency connector for support of a range of accessibility standards such as PCI-Express or Infiniband. Additionally, Real Time Operating System (RTOS) development is supported by two IBM microprocessors embedded in the FPGA, and complemented by 512MB of NAND Flash memory to provide an extremely powerful hardware/software co-design fast-prototyping and development platform. Processing power can be readily increased using additional modules in parallel, connected by a data pipe across any of the high-bandwidth interfaces previously discussed.

The entire PCB platform occupies a footprint of only 150mm x 80mm, allowing up to 8 modules to be accommodated in a traditional 1U rack-mount enclosure; an aggregate bandwidth capacity of greater than 80Gbps.

Chapter 3
Forward Error Correction

3.1 Introduction

There are many reasons why one may wish to encode data. These include security [30], where one may wish to encrypt data to prevent attack from an unauthorised third party; efficiency, where one may wish to optimise use of the available channel capacity using techniques such as *Huffman* encoding [31]; or reliability, where one seeks to counter the effects of noise on the transmission channel. Forward Error Correction (FEC) [32] is a technique which falls into the last category. FEC adds additional information in real time to the original information to be transmitted over a channel such that at the receiver, the combination of the original information and the additional information can correct errors in real time.

There are two types of codes in common use for error correction and error control; block codes and convolutional codes [32, 33]. The encoder for a block code divides the information up into message blocks of k information bits[1] each. There are thus a total of 2^k different possible messages. The encoder transforms each message into a codeword of n discrete bits. Thus, the notation (n, k) is often used to describe a code. The encoder for a convolutional code also accepts k-bit blocks and produces an encoded sequence of n bits. However, in this case each encoded block depends on both the current message word and m previous message words.

The codes considered in this research are all block codes, which find practical application in a wide range of storage and communications equipment. Of particular interest will be a special group known as the Reed-Solomon codes [34], which have been specified for use by the International Telecommunication Union in its recommendations for the Optical Transport Network [11].

[1] Assuming, for now, a binary code.

3.2 Finite field theory

The algebraic framework on which error correcting codes are based is that of finite fields[2]. The properties of such fields have some remarkable consequences for the theory and application of coding techniques. The basic taxonomy of field theory [32, 35] is introduced here.

3.2.1 Groups

Let ***G*** be a set of elements. An arbitrary binary operation $*$ on ***G*** is one which assigns to any pair of elements *a* and *b*, a uniquely defined third element $c = a * b$ in ***G***. When such a binary operation exists on ***G***, then ***G*** is closed under $*$. For example, if ***G*** is the set of all integers, then ***G*** is closed under real addition. Every real addition of two integers results in a third integer.

A set ***G*** on which a binary operation $*$ is defined is called a group if the following conditions are satisfied:

- The binary operation is associative, such that $a * (b * c) = (a * b) * c$
- ***G*** contains an identity element *e*, such that $a * e = e * a = a$
- Any element *a* in ***G*** has an inverse, such that $a * a' = a' * a = e$

The number of elements in the group defines its *order*. The set of all integers is thus a group of infinite order. Of more practical interest are *finite groups*, with a finite number of elements. The group definition outlined above may be expanded to introduce the algebraic system which is of most interest to coding theorists - the *field*.

3.2.2 Fields

A field ***F*** is a set of elements with two operations, addition $(+)$ and multiplication $(\cdot)$ satisfying the following properties:

- ***F*** is closed under $+$ and $\cdot$ such that $(a + b)$ and $(a \cdot b)$ are in ***F*** whenever *a* and *b* are in ***F***.
- The operations are commutative, such that $a + b = b + a$ and $a \cdot b = b \cdot a$.

[2]Or Galois Fields, after Evariste Galois their discoverer.

- The operations are associative, such that $(a+b)+c = a+(b+c)$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
- The operations are distributive, such that $a \cdot (b + c) = a \cdot b + a \cdot c$.

Furthermore, the additive identity element 0 and the multiplicative identity element 1 must exist in ***F*** satisfying the following:

- $a + 0 = a$ for all *a* in ***F***.
- $a \cdot 1 = a$ for all *a* in ***F***.
- For any *a* in ***F***, there exists an additive inverse element $-a$ in ***F*** such that $a + (-a) = 0$.
- For any nonzero element *a* in ***F***, there exists a multiplicative inverse element a^{-1} in ***F*** such that $a \cdot a^{-1} = 1$.

A field with a finite number of elements is commonly known as a finite, or Galois Field. The simplest example is the binary field of elements [0, 1] often denoted by $GF(2)$, and closed under modulo-2 addition and modulo-2 multiplication as shown in Figure 3.1.

Modulo-2 Addition

+	0	1
0	0	1
1	1	0

Modulo-2 Multiplication

·	0	1
0	0	0
1	0	1

Figure 3.1: *Simple modulo-2 operations*

Galois' fundamental result [35] which allows one to work with generalised finite fields beyond the restrictions of the binary case is stated without proof as follows:

Theorem 1. If and only if *q* is a prime power (i.e. $q = p^h$, where *p* is prime and *h* is a positive integer), then there exists a field $GF(q)$ of order *q*. Furthermore, if *q* is a prime power, then there is only one field of that order.

Additionally for any prime power *q*, and any positive integer *m* there exists a finite field $GF(q^m)$, called the extension field of $GF(q)$. Such extension fields are the platform on which many practical coding applications are constructed. In general a code could be constructed from any Galois Field $GF(q)$, but in practice codes with symbols from $GF(2)$ or its extension

fields $GF(2^m)$ are most widely used, since they represent an elegant mapping to the binary world of digital transmission systems. As a precursor to considering the construction of such extension fields, the concept of *irreducible* and *primitive* polynomials must be introduced.

3.2.3 Polynomials over Galois Fields

As will become apparent, it is frequently useful in the analysis of error correcting codes to consider computations with polynomials whose coefficients are from Galois Fields. For example, a polynomial $f(x)$ of degree *n*, with coefficients from the binary field $GF(2)$ could be written as:

$$f(x) = f_0 + f_1 x + f_2 x^2 + \cdots\cdots + f_n x^n \tag{3.1}$$

where each coefficient f_i is 1 or 0, the elements of $GF(2)$. Polynomial arithmetic over Galois Fields follows the commutative, associative and distributive rules of traditional polynomial arithmetic, with the addition, subtraction, multiplication and division of coefficients becoming modulo-*q* operations, where *q* is the order of the Galois Field. Polynomials therefore provide a convenient and familiar framework for the analysis of error correcting codes.

3.2.3.1 Irreducible Polynomials

A polynomial $p(x)$ over $GF(2)$ of degree *m* is said to be irreducible over $GF(2)$ if it is not divisible by any polynomial over $GF(2)$ of degree less than *m* but greater than zero. Considering the simplest examples again, of the four polynomials of degree 2; x^2, $x^2 + 1$ and $x^2 + x$ are not irreducible since they are either divisible by *x* or $x + 1$. However $x^2 + x + 1$ is not divisible by any polynomial of degree 1, and is thus irreducible.

3.2.3.2 Primitive Polynomials

An irreducible polynomial $p(x)$ of degree *m* is said to be primitive if the smallest positive integer *n* for which $p(x)$ divides $x^n + 1$ is $n = 2^m - 1$. It is difficult to distinguish primitive polynomials by inspection, but they are of fundamental importance in the construction of Galois Fields. A key property of primitive polynomials in this context is stated without proof in

Theorem 2. A more detailed analysis of the properties of primitive polynomials is presented in [36].

Theorem 2. Any primitive polynomial over $GF(2)$ of degree *m* divides $x^{2^m-1}+1$.

3.2.4 Construction of Extension Fields Based on $GF(2)$

Beginning with the two elements 0 and 1 from $GF(2)$, and following [32] one may then define a new primitive element α, and a sequence of multiplications $(\cdot)$ to generate a corresponding infinite sequence of powers of α as follows:

$$
\begin{gathered}
0 \cdot 0 = 0, \\
0 \cdot 1 = 1 \cdot 0 = 0, \\
1 \cdot 1 = 1, \\
0 \cdot \alpha = \alpha \cdot 0 = 0, \\
1 \cdot \alpha = \alpha \cdot 1 = \alpha, \\
\alpha^2 = \alpha \cdot \alpha, \\
\alpha^3 = \alpha \cdot \alpha \cdot \alpha, \\
\ldots\ldots \\
\ldots\ldots \\
\ldots\ldots \\
\alpha^j = \alpha \cdot \alpha \cdot \alpha \cdot \alpha \ldots (j\ times) \\
\ldots\ldots \\
\ldots\ldots
\end{gathered}
$$

This leaves the infinite set of elements $F = \{0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \ldots, \alpha^j, \ldots\}$. One now imposes some conditions on the set. First the set is restricted to 2^m elements. One then defines a primitive polynomial $p(x)$ of degree *m* over $GF(2)$ and imposes the condition that $p(\alpha) = 0$.

Since $p(x)$ is primitive, it divides $x^{2^m-1}+1$ (*Theorem 2*), thus:

$$x^{2^m-1} + 1 = q(x)p(x) \tag{3.2}$$

Replacing *x* with the primitive element α in 3.2 yields:

$$\alpha^{2^m-1} + 1 = q(\alpha)p(\alpha) = 0 \tag{3.3}$$

Finally, adding 1 (*modulo-2*) to both sides of 3.3 yields:

$$\alpha^{2^m-1} = 1 \tag{3.4}$$

Under the imposed conditions, the field thus becomes finite:

$$F^* = 0, 1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2^m-2} \tag{3.5}$$

It may be readily shown [32] that the resulting field is a Galois Field. Whilst the preceding analysis appears to be rather abstract, it nonetheless yields a fundamental and very useful practical conclusion. That under the condition where $p(\alpha) = 0$, one may use a primitive polynomial of degree *m* over $GF(2)$ to generate an extension field $GF(2^m)$.

Example

Consider the generation of a simple extension field $GF(2^3)$ using a primitive polynomial $p(x) = 1 + x + x^3$. Setting $p(\alpha) = 0$ yields $1 + \alpha + \alpha^3 = 0$ and thus (*modulo-2*) $\alpha^3 = 1 + \alpha$ which can be used to generate the higher powers of the field. Various standard representations for $GF(2^3)$ are shown in Figure 3.2. The required sequence of multiplications to generate the field is as follows:

$$\begin{gathered}
0 \cdot 1 = 1 \cdot 0 = 0, \\
1 \cdot 1 = 1, \\
0 \cdot \alpha = \alpha \cdot 0 = 0, \\
1 \cdot \alpha = \alpha \cdot 1 = \alpha, \\
\alpha^2 = \alpha \cdot \alpha, \\
\alpha^3 = 1 + \alpha, \\
\alpha^4 = \alpha^3 \cdot \alpha = (1 + \alpha) \cdot \alpha = \alpha + \alpha^2, \\
\alpha^5 = \alpha^4 \cdot \alpha = (\alpha + \alpha^2) \cdot \alpha = \alpha^2 + \alpha^3 = 1 + \alpha + \alpha^2, \\
\alpha^6 = \alpha^5 \cdot \alpha = (\alpha^2 + \alpha^3) \cdot \alpha = \alpha^3 + \alpha^4 = 1 + \alpha + \alpha + \alpha^2 = 1 + \alpha^2.
\end{gathered}$$

Power Representation	Polynomial Representation	N-tuple Representation
0	0	000
1	1	100
α	α	010
α^2	α^2	001
α^3	$1 + \alpha$	110
α^4	$\alpha + \alpha^2$	011
α^5	$1 + \alpha + \alpha^2$	111
α^6	$1 + \alpha^2$	101

Figure 3.2: *Power, polynomial and n-tuple representations of* $GF(2^3)$

The n-tuple representation of the field elements is simply a shorthand version of the polynomial representation. If a coefficient in the polynomial representation is 1, then the corresponding bit in the n-tuple is set. For example $1 + \alpha$ corresponds to 110, and $1 + \alpha^2$ corresponds to 101. All three representations for Galois Fields are widely used throughout the literature. Polynomial or n-tuple representations are more convenient for considering Galois Field addition or subtraction[3], since these are simple modulo-2 operations on each power of α. For example:

$$\alpha^4 + \alpha^5 = 011 + 111 = 011 \; xor \; 111 = 100 = \alpha^0 = 1 \tag{3.6}$$

In contrast, the power representation is often more convenient when considering Galois Field multiplication or division[4], since these operations may be computed by addition of indices [32]. For example:

$$\alpha^2 + \alpha = \alpha^{(2+1)} = \alpha^3 = 1 + \alpha = 110 \tag{3.7}$$

Two specific extension fields of $GF(2)$ will be of particular interest in the analysis which follows. The first of these, $GF(2^4)$, is extremely useful for prototyping and testing coding applications since calculations are of a manageable size, and the properties of $GF(2^4)$ are equally applicable to larger fields. Additionally of interest will be $GF(2^8)$, from which the Optical Transport Network Reed-Solomon codes are derived.

[3] Modulo-2 addition and subtraction are identical operations.

[4] Galois Field division is equivalent to multiplication by the inverse of the divisor.

The complete fields $GF(2^8)$ generated by primitive polynomial $p(x) = 1 + x^2 + x^3 + x^4 + x^8$, and $GF(2^4)$ generated by primitive polynomial $p(x) = 1 + x + x^4$ are given in [32].

3.3 Coding Overview

Some basic properties of the block codes around which this research is based are now introduced. In introducing these concepts it is easier to restrict discussion initially to binary codes; the key properties of such codes scale easily to the more powerful non-binary equivalents, which will be considered in due course. Assume initially that the communications system to be considered follows the traditional structure shown in Figure 3.3.

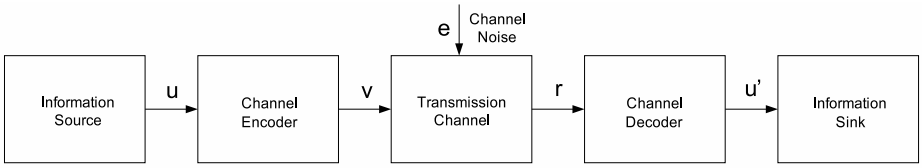

Figure 3.3: *A basic communications system*

In block coding, the binary information sequence from the source is encoded into *message* blocks of fixed length *k*, usually denoted by **u**. The encoder then adds additional information to the message blocks to produce a *transmitted codeword* of length $n(> k)$, usually denoted by **v**. This additional information (sometimes described as the *parity* bits or *overhead*) can then be used to detect and correct errors in the *received codeword* **r**, which may arise due to degradation of the channel signal-to-noise ratio. Such errors are typically represented as an *error vector* **e**. If the gain of the chosen coding scheme is sufficient to reverse the effect of the channel noise, the decoded message **u'** will be identical to the original information sequence **u**.

3.3.1 Useful Properties of Block Codes

Consider the classic single-error-correcting code example shown in Figure 3.4. This is a $(7, 4)$ Hamming code from the first set of error correcting codes to be developed [37].

This code has a number of key characteristics which make it extremely attractive for implementation in hardware. Although the example shown is trivial, its key properties also apply to the more complex, more powerful codes deployed in many real world applications.

Message	Codeword	Message	Codeword
0001	1010001	0000	0000000
1001	0111001	1000	1101000
0101	1100101	0100	0110100
1101	0001101	1100	1011100
0011	0100011	0010	1110010
1011	1001011	1010	0011010
0111	0010111	0110	1000110
1111	1111111	1110	0101110

Figure 3.4: *The Hamming* $(7, 4)$ *block code*

The code is *linear* - the modulo-2 sum of any two codewords is also a codeword; *systematic* - the message word is present unaltered in the right-most digits of the codeword; and *cyclic* - any cyclic shift (left or right) of any codeword is itself a codeword.

These properties map intuitively to hardware synthesis, particularly to well-defined structures such as shift registers and simple combinatorial circuits (a modulo-2 adder is simply an xor gate, for example). Techniques applicable to linear, systematic, cyclic codes are thus the focus of this research.

3.3.1.1 The Generator Polynomial

One of the fundamental characteristics of cyclic codes is the existence of a unique codeword around which the entire code can be constructed. The significance of this polynomial is encapsulated in two key theorems (again stated without proof) below.

An excellent treatment of polynomial arithmetic in the context of cyclic codes is given in [32]. Discussion here is limited to that required to clarify nomenclature used in the analysis of encoding and decoding techniques which follows.

Theorem 3. For an (n, k) cyclic code, C, there exists a nonzero code polynomial of minimum degree $n - k$ which is unique.

Theorem 4. Following from *Theorem 3*, if $g(x)$ is this nonzero code polynomial of minimum degree $n - k$ in an (n, k) cyclic code C, then any other binary polynomial[5] is a codeword in C if and only if it is a multiple of $g(x)$.

Information messages and codewords are routinely represented using polynomial arithmetic,

[5] A polynomial with coefficients from $GF(2)$.

identical to that used to describe Galois Field elements. Typically, the message to be encoded **u** is given by $u(x)$ and the resultant codeword **v** by $v(x)$. For example a binary codeword **v** = 10101010 is the same as $v(x) = x^7 + x^5 + x^3 + x$, a message word **u** = 0111 is the same as $u(x) = x^2 + x + 1$.

With this convention in mind, and following on from *Theorem 3* and *Theorem 4*, it may be noted that every code polynomial $v(x)$ in an (n, k) cyclic code can be expressed in the following form:

$$v(x) = u(x)g(x) = (u_0 + u_1x + \ldots + u_{k-1}x^{k-1})g(x) \tag{3.8}$$

where $u(x)$ is the message to be encoded and $v(x)$ is the resultant codeword. A cyclic code C is therefore completely specified by the nonzero polynomial of minimum degree, $g(x)$, which is thus termed the *generator polynomial*, and often presented in one of the following shorthand formats:

$$g(x) = \prod_{i=0}^{2t-1}(x - \alpha^{j_0+i}) = \sum_{i=0}^{2t} g_i x^i \tag{3.9}$$

3.3.1.2 Minimum Distance and Error-Correcting Capability

A final fundamental property of block codes, which is directly related to their error-correcting capability, is the *minimum distance*. The Hamming Distance between any two codewords is defined as the number of places in which those codewords differ.

The minimum distance of a code C is defined as the smallest of the Hamming distances between any two distinct codewords in the code:

$$d_{min}(C) = min\,\{d(x, y)\}\ \ x, y \in C, x \neq y \tag{3.10}$$

Example

If C is a simple binary code as shown in Figure 3.5, then the Hamming distance between c_1 and c_2 is 3, since they differ in 3 positions. In fact, for this example the Hamming distance between any two distinct codewords is 3. Thus the minimum distance of C is 3. It may be

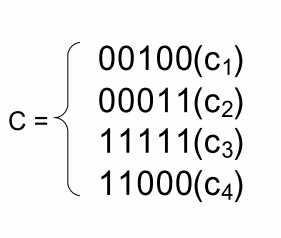

Figure 3.5: *A trivial binary code*

shown [32, 36] that the number of errors which a code can detect and correct, *t* is directly related to its minimum distance as follows:

$$t = \frac{1}{2}(d_{min} - 1) \tag{3.11}$$

This error-correcting capability completes the taxonomy required to formally and completely describe a binary linear block code. For example, for any positive integer $m \geq 2$, there exists a Hamming code with the following characteristics:

- Block length: $n = 2^m - 1$
- Information symbols: $k = 2^m - m - 1$
- Parity symbols: $n - k = m$
- Minimum distance: $d_{min} = 3$
- Error-correcting capability $= 1$

3.3.2 From Single to Multiple-Error-Correcting Codes

Clearly the scope of practical application for a single-error-correcting code is limited. Fortunately, there exists an elegant generalisation of the Hamming codes for correcting multiple errors, discovered by Bose, Chaudhuri [38] and independently by Hocquenghem [39], and thus known as the BCH codes, defined as follows.

For any positive integers $m(\geq 3)$ and $t(< 2^{m-1})$, there exists a binary BCH code with the following characteristics:

- Block Length: $n = 2^m - 1$
- Parity Symbols: $n - k \leq mt$
- Minimum Distance: $d_{min} \geq 2t + 1$

This code is capable of correcting any combination of t or fewer errors in a block of n digits, and is thus called a *t-error-correcting* code. Recalling that α is defined as a generalised primitive element in $GF(2^m)$, the generator polynomial $g(x)$ of a BCH code is then defined as the lowest degree polynomial over $GF(2)$ which has the powers of α up to α^{2t} as its roots - that is $g(\alpha^i) = 0$ for $1 \leq i \leq 2t$.

3.3.3 From Binary to Non-binary Codes

The simple Hamming code in Figure 3.4 is an example of a binary block code. It uses one bit per code symbol, and can correct 3 bit errors. Furthermore, the binary BCH codes offer a useful and elegant extension of the Hamming codes. However, such codes remain of limited practical use over high bandwidth optical transmission systems where bit error rate (BER) thresholds are extremely demanding.

3.3.3.1 Extending the BCH Codes

To meet these exacting standards, the BCH codes may be further extended by using multiple bits per symbol, rather than just one, to dramatically increase error correction and detection ability. In fact, if p is a prime number and q is any power of p, then there exist codes from the Galois Field $GF(q)$, called the *q-ary* codes, with q bits per symbol.

The Reed-Solomon codes [34], named after their discoverers, are an example of such codes, and represent a key non-binary sub-class of the BCH codes. Reed-Solomon codes have been widely adopted in modern communications systems, despite the fact that at the time of their discovery, no practical applications were known. The RS(255,239) code is of particular relevance to this research, being the code standardised for use in the Optical Transport Network. RS(255,239) uses 8 bits per code symbol, and can correct 8 symbol errors. The advantage of this is best demonstrated by example.

Consider a stream of 8 code symbols from the RS(255,239) code, with one bit error per symbol

as shown in Figure 3.6. 8 bits are in error, thus 8 symbols are in error and all the bit errors can be corrected; x denotes a bit error.

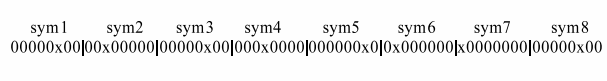

Figure 3.6: *Non binary code with 8 symbol errors - 1 bit error per symbol*

Now consider a stream of 8 symbols from the same code in Figure 3.6, with all the bits in error.

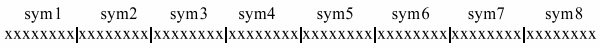

Figure 3.7: *Non binary code 8 symbol errors - 8 bit errors per symbol*

Although there are now 64 contiguous bits in error, there are still only 8 symbol errors, so all of the erroneous bits can still be corrected. This ability to deal with long runs of contiguous error bits is extremely useful in communications networks, where intermittent faults may lead to bursty error profiles.

3.3.3.2 The Reed-Solomon Codes Defined

For any choice of positive integers s and t, there exists a $q-ary$ BCH code of length $n = q^s - 1$, which is capable of correcting any combination of t or fewer errors and requires no more than $2st$ parity-check digits. With $q = 2$, one again obtains the binary BCH codes described previously. The Reed-Solomon codes are the subclass of BCH codes for which $s = 1$. In general, a *t-error-correcting* Reed-Solomon code with symbols from $GF(q)$ has the following properties:

- Block Length: $n = q - 1$
- Parity Symbols: $n - k = 2t$
- Minimum Distance: $d_{min} = 2t + 1$

RS(255,239) is thus a code with symbols from $GF(256)$, requiring 8 bits per symbol for distinct representation of each field element. A complete table of $GF(256)$ elements, as generated by primitive polynomial $p(x) = 1 + x^2 + x^3 + x^4 + x^8$ is given in [32]. Each information block

for encoding is 239 symbols long, with a resultant codeword 255 symbols long. The code is therefore capable of detecting and correcting 8 symbol errors, as illustrated in 3.3.3.1.

In considering the arithmetic and manipulation of non-binary codes it is common, as with the binary case, to use polynomial notation. In the simple example of binary codes, the polynomials take their coefficients from $GF(2)$; every coefficient is either a 1 or a 0. For the non-binary codes, the polynomials take their coefficients from $GF(q)$. These coefficients are usually defined by their power representation, as in the examples shown below:

$$u(x) = \alpha^{24}x + \alpha^{6}x^{2} + \alpha^{132}x^{3} + \ldots\ldots + \alpha^{3}x^{239} \tag{3.12}$$

$$v(x) = \alpha^{16}x + \alpha^{210}x^{3} + \alpha^{88}x^{4} + \ldots\ldots + \alpha^{16}x^{255} \tag{3.13}$$

3.4 Forward Error Correction in the Optical Transport Network

In addition to the network management functions mentioned in 2.2 the G.709 Optical Transport Network also specifies capacity for FEC as shown in the Optical Transport Unit (OTU) frame structure of Figure 3.8. Byte numbers are annotated.

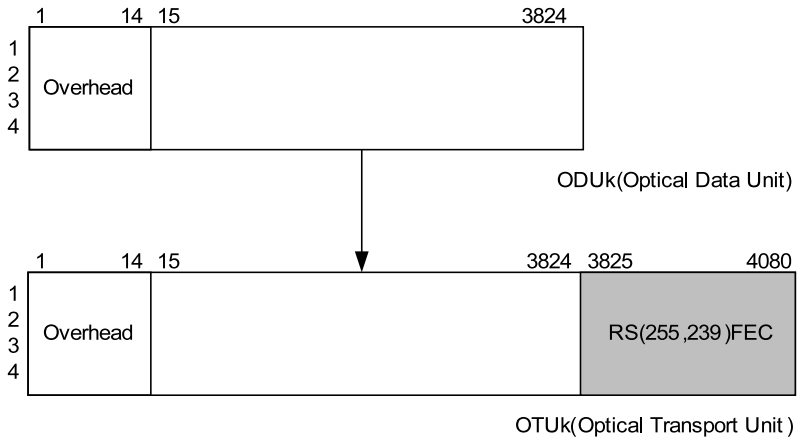

Figure 3.8: *OTUk frame structure*

Each of the OTUk[6] sub-rows is 4080 bytes long, and may be subdivided into 16 interleaved 255 byte Reed-Solomon calculations as shown in Figure 3.9. The first byte in OTU row 1 is

[6] *k* refers simply to line rate, see Table 3.1.

mapped into first byte in sub-row 1, the second into the first byte in sub-row 2 and so on until the sixteenth byte is mapped into the first byte in sub-row 16. The process then wraps around such that the seventeenth byte from OTU row 1 is mapped into the second byte in sub-row 1, the eighteenth into the second byte in sub-row 2 and so on. Each of the four OTU rows is constructed and deconstructed in this fashion.

This interleaving significantly enhances the burst-error performance of the code. Recall that the non-interleaved RS(255,239) code may detect and correct up to 8 symbol errors, or 64 contiguous bit errors. For this to occur in any of the sub-row FEC calculations, 1024 contiguous bits would have to be in error in the OTU*k* frame structure, since the interleaving distributes these errors evenly between the sub-rows. Thus the interleaved code can detect and correct up to 1024 contiguous errors in an OTU*k* frame.

In wireline transmission systems this corresponds to an effective coding gain of 7 to 8dB. For very high bandwidth optical networks with transmission speeds up to 43Gbps this acts to counter signal-to-noise ratio degradation due to fibre loss, chromatic dispersion and other aberrations.

Effectively this means that network operators can achieve improved transmission distance without increasing optical power, or improved bit error rate margins across the existing network span.

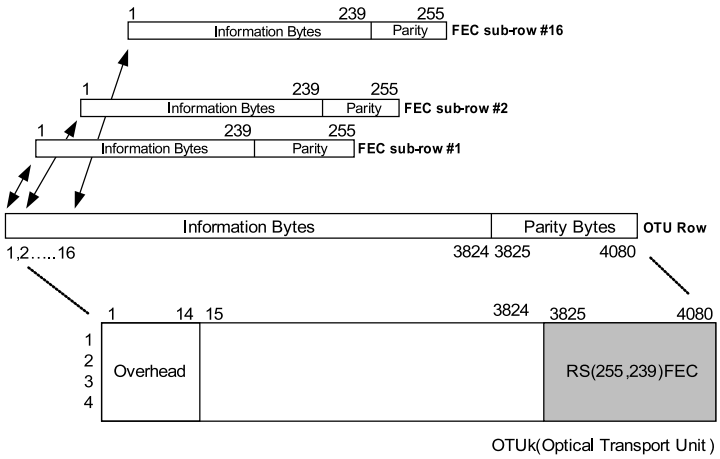

Figure 3.9: *Byte interleaved FEC as specified by ITU-T G.709*

The frame structure defined for the Optical Transport Network is fixed and independent of line rate. At higher speeds, the period of transmission for each frame decreases. The

Type	Nominal Bit Rate	Line Rate (Mbps)	Frame Period (µs)
OTU-1	255/238 * OC-48	2,666.06	48.971
OTU-2	255/237 * OC-192	10,709.23	12.191
OTU-3	255/236 * OC-768	43,018.41	3.035

Table 3.1: *OTU type and capacity*

currently defined transmission rates are based on OC-48, OC-192 and OC-768, each scaled for the additional FEC overhead. Note that the fractional scaling of these rates is not exactly 255/239, since additional bytes (unrelated to the FEC functionality and sometimes referred to as "stuffing" bytes) are added when mapping the client signal overhead [11].

3.5 Forward Error Correction for 43Gbps Systems

The research detailed here focuses on OTU-3 systems, running at approximately 43Gbps. Such systems present a number of unique challenges to both electronic system designers and optical network providers, given the speeds which must be accommodated for real time processing and network monitoring.

Allowing adequate margin for system jitter, the current target FPGA silicon is capable of running at approximately 170MHz [24, 40]. This means that on-chip interfaces need to be 256 bits (32 bytes) wide. This processing parallelism maps directly to silicon area - the wider the data interface the larger the implementation. This in turn presents a challenge for the high-speed designs typical of communications systems, as routing delays and the layers of logic required for multi-bit processing make it difficult to achieve timing closure.

Additionally, since there are 16 interleaved Reed-Solomon calculations in every OTU*k* row, processing data 32 bytes wide delivers two bytes from each Reed-Solomon calculation in every clock cycle, which adds further to the need for parallel processing engines and control hardware to accommodate the line rate. This research addresses all of these issues in investigating ways to gracefully process FEC in 43Gbps systems, in the context of both encoding and decoding architectures.

3.5.1 Reed-Solomon Encoding

If $u(x)$ is an arbitrary message polynomial, it may be shown [32, 36] that there are three steps to systematically encoding $u(x)$ to produce a corresponding codeword in C, $v(x)$. These steps are as follows:

- Premultiply the message polynomial $u(x)$ by x^{n-k}
- Divide the product $u(x)x^{n-k}$ by $g(x)$, the generator polynomial of C, to obtain the remainder polynomial $b(x)$.
- Form the codeword $v(x) = b(x) + x^{n-k}u(x)$

Whilst these operations at first appear convoluted, they map elegantly to a linear feedback shift-register (LFSR) structure. Premultiplication of the message polynomial is a simple shift operation, whilst Peterson and Weldon [41] have presented a detailed treatment of how polynomial long division with a fixed divisor maps to an LFSR, such as that shown in Figure 3.10.

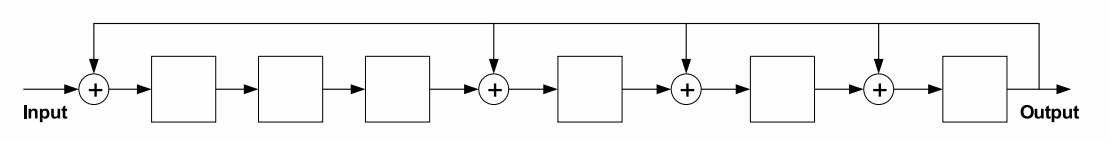

Figure 3.10: *Circuit for dividing* $x^6 + x^5 + x^4 + x^3 + 1$

The LFSR structure is readily extended to polynomials with coefficients from extension fields of $GF(2)$. The non-zero feedback taps must now become Galois Field multipliers, the summation blocks must become Galois Field adders and the storage elements must scale to store multiple bits per symbol. Otherwise, the steps involved in systematic encoding of Reed-Solomon codewords are identical to the binary operations.

The architecture for a standard division circuit RS(255,239) encoder is shown in Figure 3.11. The message symbols to be encoded are shifted into the LFSR as shown (with the gate turned on) and simultaneously shifted onto the transmission channel. The message symbols are thus multiplied by feedback coefficients corresponding to the generator polynomial. Once all the message symbols have been shifted in, all the Reed-Solomon parity digits have been calculated

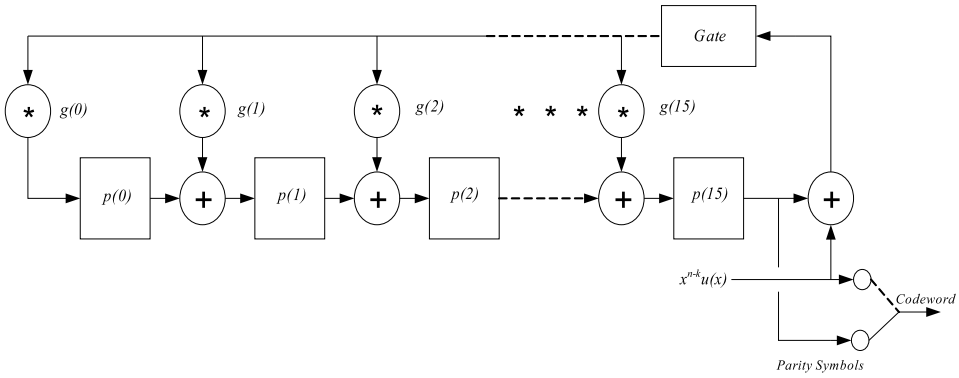

Figure 3.11: *Standard $RS(255, 239)$ encoder architecture*

and are stored in the registers. The feedback path is broken by turning off the gate and the parity digits shifted onto the transmission channel to complete the codeword.

Given the relative simplicity of the traditional Reed-Solomon encoder structure of Figure 3.11, previous research in this field has been biased towards optimising the performance of the arithmetic units rather than the architecture itself, which for most applications performs optimally. In particular the optimisation [25, 42, 43] and synthesis [44] of finite field multipliers, which represent the computational bottleneck and thus the critical timing path in any Reed-Solomon implementation, have both been afforded significant effort. Other areas of recent research interest include reconfigurable encoders [45] which can be dynamically programmed to implement different codes or improve efficiency, and rate adaptive variants [46].

The research detailed here maintains a more architectural focus. At the time of investigation, very high performance Galois Field arithmetic blocks optimised for FPGA hardware had already been developed by Aliathon Ltd. Furthermore, investigation into dynamic reconfiguration or rate adaptability was not deemed appropriate given the fixed nature of G.709 encoding. In considering this fixed encoder structure, the OTU-3 specific case presents a number of unique design challenges which this research seeks to address.

3.5.2 A New Approach to Reed-Solomon Encoding

The structure presented in Figure 3.11 is intrinsically a single-symbol processing engine; only one symbol at a time may be fed into the shift-register. This presents difficulties at

OTU-3 transmission speeds. Given the limitations of silicon speed, the on-chip interfaces for processing 43Gbps optical rates are 32 bytes wide. As previously illustrated, each sub-row in the OTU-3 frame comprises 16 interleaved FEC calculations. Thus, on every system clock cycle, two bytes from each FEC calculation are delivered and must be processed in real time.

Consider the operation of an OTU-3 sub-row calculation based on a single-symbol encoder as illustrated in Figure 3.12. For simplicity, the figure shows small input sub-rows of 8 symbols each, labelled alphabetically. In reality, the incoming messages will always comprise an odd number of symbols; a significant point which will be discussed again in due course.

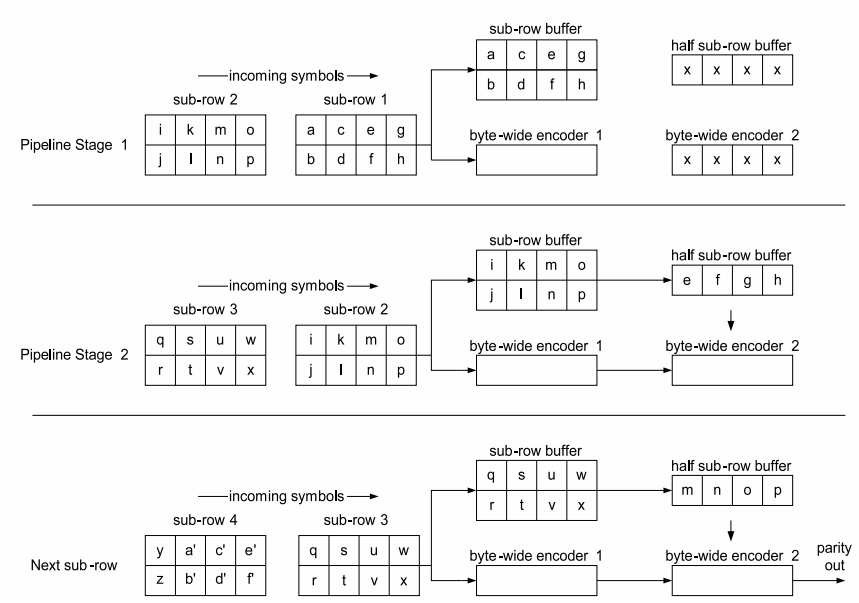

Figure 3.12: *OTU-3 encoding based on single symbol encoders*

To reiterate the fundamental difficulty in using a single-symbol engine - in every clock cycle 2 input symbols arrive and only one may be processed. Thus to keep up with the incoming data, one must introduced parallel encoders, data buffers and pipelined logic. In the current example the first sub-row to arrive contains input symbols *a,b,c,d,e,f,g,h*. These symbols arrive in four clock cycles during pipeline stage 1. During this processing window, all 8 symbols are written to a sub-row buffer, and symbols *a,b,c* and *d* are fed to the first encoder.

In pipeline stage 2, the next sub-row arrives comprising symbols *i,j,k,l,m,n,o,p*. As before these 8 symbols are written to the sub-row buffer and symbols *i,j,k* and *l* are fed to the first encoder. Also in pipeline stage 2, since symbols *e,f,g* and *h* from the first sub-row to arrive have not yet been processed, these are fed to a second buffer and processed by the second encoder. This

second encoder must be initialised with the parity values produced by the first encoder during pipeline stage 1 for the stage 2 calculation to be valid. At the end of pipeline stage two, the parity symbols associated with the first sub-row to arrive are valid. Subsequent sub-rows are processed in identical 2-stage fashion.

An OTU-3 encoder core based on the architecture shown in Figure 3.11 would thus require 32 LFSR structures in parallel, 16 full sub-row buffers and 16 half sub-row buffers, the buffers utilising 32 embedded block-RAMs. The architecture also introduces a sub-row's worth of additional latency and requires relatively complex pipelining and control logic.

Clearly a reformulation of the traditional Reed-Solomon encoder capable of processing two symbols per clock cycle would be useful in addressing this complexity. Fredrickson [47] has followed this approach in formulating a word-wise encoder limited to generating two parity symbols, based on an analysis of the behaviour of a traditional encoder over two clock cycles.

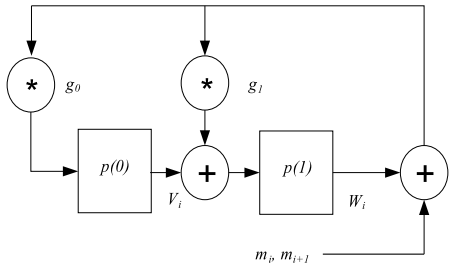

Figure 3.13: *Simple encoder for analysis over two clock cycles*

For the simple example given, assuming that the message symbol inputs to the encoder are m_i and m_{i+1} respectively, Fredrickson notes that the contents of the registers $p(0)$ and $p(1)$ after two clock cycles may be given by the following equations:

$$p(0) = g_0 g_1(w_i + m_i) + g_0(v_i + m_{i+1}) \tag{3.14}$$

$$p(1) = g_1^2(w_i + m_i) + g_1(v_i + m_{i+1}) + g_0(w_i + m_i) \tag{3.15}$$

Additionally, Fredrickson has noted that the introduction of a fixed offset for all the powers of α in any field-specific encoder realisation can be used to reduce the number of Galois Field multipliers required.

Recall the generalised expression for the code generator polynomial was given by Equation 3.9 as:

$$g(x) = \prod_{i=0}^{2t-1}(x - \alpha^{j_0+i}) = \sum_{i=0}^{2t} g_i x^i$$

For the encoder of Figure 3.13, defined for a single-error-correcting code with symbols from $GF(256)$ the generator can be expanded as follows:

$$g(x) = (x - \alpha^{j_0})(x - \alpha^{j_0+1}) = x^2 - (\alpha^{j_0} + \alpha^{j_0+1})x + \alpha^{2j_0+1} \tag{3.16}$$

Hence the individual generator coefficients corresponding to the LFSR feedback taps may be given by:

$$g_2 = 1;\;\; g_1 = \alpha^{j_0} + \alpha^{j_0+1};\;\; g_0 = \alpha^{2j_0+1} \tag{3.17}$$

Noting that the field for this example is $GF(256)$, the fixed offset j_0 can be chosen such that either g_1 or g_0 equal 1, thus removing the Galois Field multiplier at that stage in the LFSR circuit. Thus in one possible embodiment with $j_0 = 127$ the g_0 multiplier is eliminated. In the other possible embodiment with $j_0 = 230$ the g_1 multiplier is eliminated. Fredrickson demonstrates by inspection that the latter yields a more efficient hardware solution, with a critical path traversing two Galois Field adders and one Galois Field multiplier as shown in Figure 3.14.

Whilst this field-specific optimisation technique is relatively easy to apply to the encoder example with a simple quadratic generator polynomial specified by [47], it becomes more difficult to apply to larger generators where the coefficients comprise multiple terms in α. Furthermore, identifying which coefficient elimination will result in an optimal hardware solution becomes a non-trivial task, and the resultant architecture does not map intuitively to the traditional encoder structure of Figure 3.11.

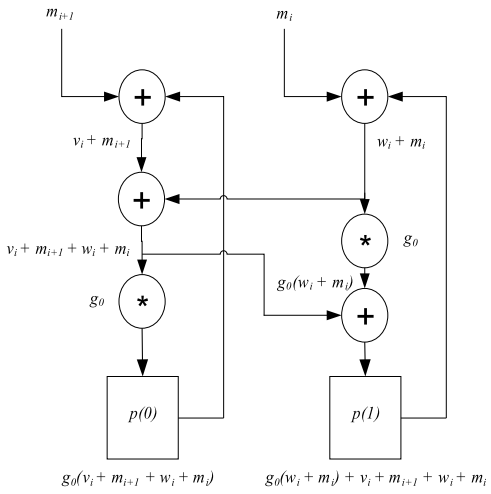

Figure 3.14: *Reformulated two-symbol encoder for* $g(x) = x^2 - x + \alpha^{206}$

3.5.3 A Modified Reed-Solomon Encoder for OTU-3

An alternative two-symbol Reed-Solomon encoder is proposed here. The starting point for development of the new encoder is similar to that adopted by Fredrickson - analysis of the behaviour of a traditional encoder system over two clock cycles. By deriving a new generalised expression relating the parity symbol values to two input symbols rather than one, it is possible to construct an encoder with a guaranteed optimal critical path delay which is independent of the Galois Field used and the size of the generator polynomial. The new approach is thus inherently scalable and does not require any offset to be introduced in the Galois Field arithmetic.

The resulting architecture is more complex, in the sense that additional parallel multiplications are introduced. However the new feedback coefficients remain constants which may be pre-computed. Since the newly introduced multiplications are in parallel with the original generator feedback taps, the critical path of the new encoder is the same as its predecessor, traversing two adders and one multiplier. Consider the upper segment of a generalised encoder structure as shown in Figure 3.15.

Here, $p(m)$ represents the upper-most parity symbol in the code, and $p(n)$ is an arbitrary parity digit. Message m_0 is shifted into the encoder first, followed by message m_1. After one clock cycle, the value stored in $p(n)$ is given by:

$$p(n)_0 = g_n(p(m)_0 + m_0) + p(n-1)_0 \tag{3.18}$$

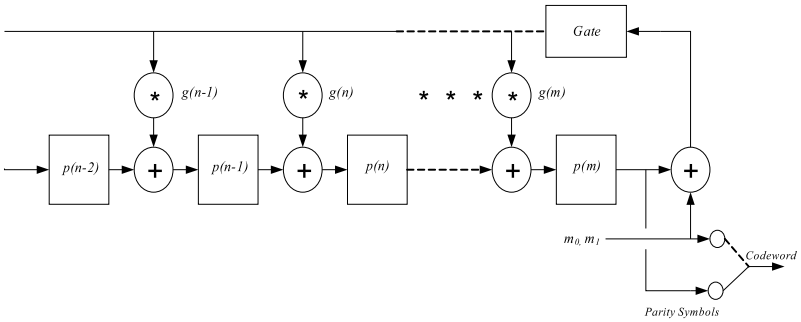

Figure 3.15: *Most significant symbols of a Reed-solomon encoder*

After a further clock cycle, message m_1 has been shifted into the encoder, and the value stored in $p(n)$ is given by:

$$p(n)_1 = g_n(p(m)_1 + m_1) + p(n-1)_1 \tag{3.19}$$

This can be expanded to produce a generalised expression relating an arbitrary parity symbol to two message symbols:

$$p(n)_1 = g_n(g_m(p(m)_0 + m_0) + p(m-1)_0 + m_1) + g_{n-1}(p(m)_0 + m_0) + p(n-2)_0 \tag{3.20}$$

$$p(n)_1 = (g_n g_m + g_{n-1})(p(m)_0 + m_0) + g_n(p(m-1)_0 + m_1) + p(n-2)_0 \tag{3.21}$$

Reformulating the generalised expression to yield Equation 3.21, one can readily implement an encoder structure based on the new arithmetic to compute parity digits by processing two symbols simultaneously, as shown in Figure 3.16.

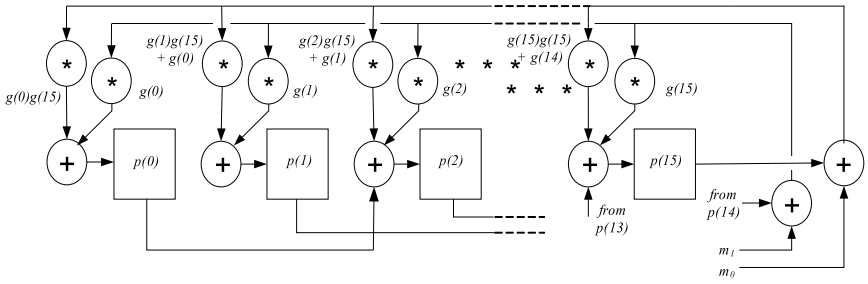

Figure 3.16: *A modified Reed-Solomon RS(255,239) encoder for OTU-3*

Having now developed an encoder capable of processing two Reed-Solomon symbols per clock cycle, there remains one further barrier to an OTU-3 compliant solution. Intrinsically, the number of symbols per message word in any Reed-Solomon code is odd. Thus, in processing such messages two symbols at a time it is necessary to deal with an odd cycle in every message word, when one of the input symbols to the encoder is non-valid, as illustrated in Figure 3.17 for a simple RS(15,9) code.

In this example, data is presented directly to the two symbol encoder from the OTU-3 line interface. For the RS(15,9) code, nine information symbols and six null parity symbols (to be replaced by the contents of the encoder registers at the end of each calculation) for each message word are presented to the encoder inputs. The symbols of the first message, denoted $m1_x$ and $p1_x$ are presented first, followed by the first symbols of the second message, denoted $m2_x$. Since the encoder processes two symbols at a time (on input channels 1 and 2), the fifteenth symbol (null parity) of the first message overlaps with the first information symbol of the second. This results in an incorrect calculation of parity symbols for the second message word.

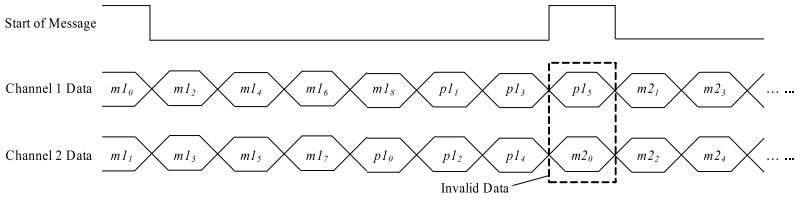

Figure 3.17: *Invalid data overlap in a two-symbol encoder*

There are a number of possible ways to deal with this. For example, with adequate buffering one could implement a standard encoder in parallel with the two symbol variant and switch to the single byte stream during the odd cycles. This would add a significant amount of latency and redundant hardware.

The solution proposed here for OTU-3 is to implement a wrapper, designed to fit around the encoder core, which formats the G.709 compliant data in such a way that the encoder always processes an even number of symbols. The wrapper does this by the insertion of a leading zero symbol (which does not change the values stored in the parity registers) during the odd cycles. Since the code is systematic, this leading zero also appears on the output data stream from the encoder. When this happens, the wrapper removes the zero symbol, so that to any device interfacing to the encoder, the data stream remains G.709 compliant.

The encoder and wrapper have been implemented in VHDL and tested using behavioural test-bench modules, with the output parity symbols verified against a known Reed-Solomon tool from Bell Labs [48]. The new encoder utilises approximately 350 slices[7] compared with approximately 170 for the single symbol variant when built for the Virtex II Pro family of FPGA devices, running at clock speeds greater than 200MHz. However, since only 16 of the new engines are required for OTU-3 (compared with 32 of the traditional engines), and no control logic is required to transfer data between cascaded encoder stages, the arithmetic logic requirements of both solutions are nominally equivalent.

The principal advantage of the new architecture is the embedded memory saving. The OTU-3 encoder proposed here completely removes the need for any interim buffering of sub-rows, thus saving 32 block-RAMs and removing the latency introduced in the single-symbol pipelined implementation.

3.5.4 Reed-Solomon Decoding

Decoding Reed-Solomon codes is arithmetically more complex than the equivalent encoding, and thus typically represents the performance bottleneck in FEC systems. The main processing blocks in a typical Reed-Solomon decoder are shown in Figure 3.18.

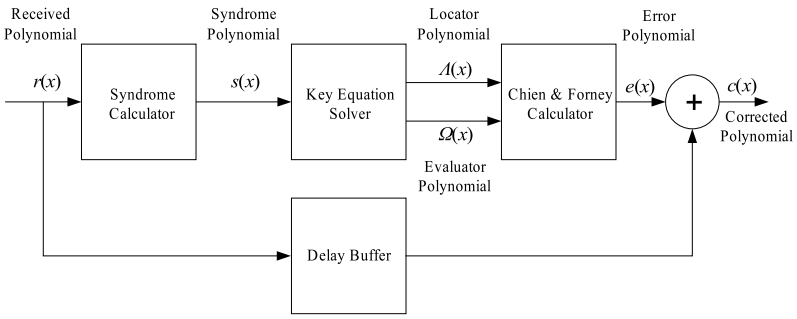

Figure 3.18: *A typical Reed-Solomon decoder system*

3.5.5 Syndrome Calculator

Calculation of a special polynomial known as the *syndrome* is an integral part of many decoding schemes for block codes. Appreciation of the significance of the syndrome polynomial requires

[7] A slice is a standard Xilinx FPGA logic block, see [24] for details.

a brief review of some further background coding theory. Some equations which will prove useful in subsequent analysis are stated here without proof. For a detailed treatment the reader is referred to [32, 35].

In the analysis of block codes it is common to use matrix notation (in addition to the polynomial representation used thus far). For example, the cyclic code generated by $g(x) = 1 + x + x^3$ has a corresponding generator matrix given by **G**:

$$\mathbf{G} = \begin{Bmatrix} 1101000 \\ 0110100 \\ 0011010 \\ 0001101 \end{Bmatrix}$$

Figure 3.19: *Generator matrix for* $g(x) = 1 + x + x^3$

This matrix **G** is generally not in systematic form, but can be reformulated by simple row operations, and presented systematically. In its general form, this matrix may be written in terms of a sub-matrix **P** and an identity matrix **I** such that $\mathbf{G} = [\mathbf{P} \;\; \mathbf{I}_k]$, as follows:

$$\mathbf{G} = \left\{ \left[\begin{array}{ccc|c} p_{0,0} & p_{0,1} & \cdots \cdots p_{0,\,n-k-1} & 1\,0\,0 \ldots 0 \\ p_{1,0} & p_{1,1} & \cdots \cdots p_{1,\,n-k-1} & 0\,1\,0 \ldots 0 \\ p_{2,0} & p_{2,1} & \cdots \cdots p_{2,\,n-k-1} & 0\,0\,1 \ldots 0 \\ & & & \\ p_{k-1,0} & p_{k-1,1} & \cdots \cdots p_{k-1,\,n-k-1} & 0\,0\,0 \ldots 1 \end{array} \right] \right\}$$

P matrix — k x k identity matrix

Figure 3.20: *A systematic generator matrix*

If the generator matrix is stated in this form, a corresponding parity check matrix also exists, given by $H = [\mathbf{I}_{n-k} \;\; \mathbf{P}^T]$ where $\mathbf{P}^T$ is the transpose of the **P** matrix shown above. Having defined the parity check matrix **H**, the syndrome is given (without proof here) as the following $(n-k)$-tuple, where $\mathbf{H}^T$ is the transpose of **H** and **r** is the received vector:

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = (\mathbf{v} + \mathbf{e}) \cdot \mathbf{H}^T = \mathbf{v} \cdot \mathbf{H}^T + \mathbf{e} \cdot \mathbf{H}^T \tag{3.22}$$

It may readily be shown that $\mathbf{v} \cdot \mathbf{H}^T = 0$. Thus the syndrome is dependent only on the received error vector **e**. This is what makes it so important as a decoding mechanism - since regardless of the transmitted codeword, one has a way of determining (within the limits of the chosen scheme) the error vector.

Furthermore, returning once again to the familiar polynomial notation, it may readily be shown [32] that the syndrome components can be related directly to the received error vector as follows (where n is the number of errors):

$$s_i = \sum_{j=0}^{n-1} e_j(\alpha^i)^j = e_0 + e_1\alpha^i + e_2(\alpha^i)^2 + \ldots\ldots + e_{n-1}(\alpha^i)^{n-1} \tag{3.23}$$

The syndrome coefficients are thus readily calculated using a simple division circuit similar in basic structure to a single encoder coefficient, shown in Figure 3.21.

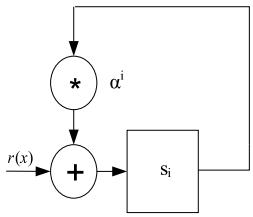

Figure 3.21: *A simple syndrome calculator circuit*

3.5.6 Key Equation Solver (KES)

The task of returning a valid error polynomial from a syndrome polynomial is an inherently difficult one. Assuming that the syndrome polynomial from the latest received codeword has been successfully computed, then:

$$s(x) = s_1 + s_2x + s_3x^2 + \ldots + s_{2t}x^{2t-1} = \sum_{i=0}^{2t-1} s_{i+1}x^i \tag{3.24}$$

From Equation 3.23 it may be noted that the syndrome coefficients themselves may be represented as a summation of received vector terms in α. Thus, substituting Equation 3.23 into Equation 3.24, the syndrome polynomial may be expressed as a double summation.

$$s(x) = \sum_{i=0}^{2t-1}\sum_{j=0}^{n-1} e_j\alpha^{(i+1)j}x^i \tag{3.25}$$

Noting that e_j is only non-zero if j is an error location $(j \in M)$ one may simplify and reorder this summation to give:

$$s(x) = \sum_{j \in M} e_j \alpha^j \sum_{i=0}^{2t-1} \alpha^{ij} x^i \tag{3.26}$$

Expanding the inner sum of Equation 3.26 fully one obtains the following geometric progression:

$$\sum_{i=0}^{2t-1} \alpha^{ij} x^i = 1 + \alpha^j x + \alpha^{2j} x^2 + \ldots\ldots + \alpha^{(2t-1)j} x^{2t-1} = \frac{1 - (\alpha^j x)^{2t}}{1 - (\alpha^j x)} \tag{3.27}$$

Substituting the sum of this geometric progression into Equation 3.26 one obtains the following:

$$s(x) = \sum_{j \in M} e_j \alpha^j \frac{1 - \alpha^{2tj} x^{2t}}{1 - \alpha^j x} \tag{3.28}$$

$$s(x) = \sum_{j \in M} \frac{e_j \alpha^j}{1 - \alpha^j x} - \sum_{j \in M} \frac{e_j \alpha^j \alpha^{2t+1} x^{2t}}{1 - \alpha^j x} = \frac{\omega(x)}{l(x)} - \frac{u(x) x^{2t}}{l(x)} \tag{3.29}$$

From Equation 3.29 it may be seen that $l(x)$ is the product of all the terms $(1 - a^j x)$ as j runs through the error locations M:

$$l(x) = \prod_{j \in M} (1 - \alpha^j x) \tag{3.30}$$

The roots of $l(x)$ are the inverses of the powers a^j for when j is an error location, thus $l(x)$ is commonly referred to as the *error locator* polynomial, since its solution yields the error locations. Therein lies the difficulty for practical systems - since Equations 3.25 through 3.30 are based on *a priori* knowledge of the error vector, which a practical real-time decoder will not have.

From Equation 3.29 one may also derive the definitions of two other significant polynomials:

$$\omega(x) = \sum_{j \in M} e_j \alpha^i \prod_{i \in M, i \neq j} (1 - \alpha^i x) \tag{3.31}$$

$$u(x) = \sum_{j \in M} e_j \alpha^{(2t+1)j} \prod_{i \in M, i \neq j} (1 - \alpha^i x) \tag{3.32}$$

Having determined the error locations using $l(x)$, one can use $\omega(x)$ to determine the error values at those locations. $\omega(x)$ is thus commonly known as the *error evaluator polynomial*. Equally, one could use $u(x)$ to determine the same error values, hence $u(x)$ is sometimes referred to as the *error co-evaluator*; of theoretical interest though not actually required in practice. Equation 3.29 is often restated without explicit reference to $u(x)$ in the form of the polynomial congruence[8] known as Berlekamp's Key Equation [49], after its discoverer. The locator is commonly denoted by $\sigma(x)$ in the literature; the notation adopted for the remainder of this analysis.

$$\sigma(x)s(x) = \omega(x) \; mod \; x^{2t} \tag{3.33}$$

This Key Equation is aptly named, being the cornerstone of BCH and Reed-Solomon encoding. The Key Equation Solver (KES) block takes the syndrome as input and produces the error locator and error evaluator polynomials as outputs. Since solution of the key equation is mathematically intractable, the KES architecture is typically the most complex in any Reed-Solomon system.

3.5.7 Chien Search and Forney Calculation

The final stage in Reed-Solomon decoding is to establish the error locations and the error values corresponding to those locations. Having established the error locator polynomial $\sigma(x)$ one seeks to find its roots, which yield the inverse error locations. Since $\sigma(x)$ is a polynomial with coefficients from a finite field, it is practical (at least for fields of manageable size) to substitute every possible field element as a root. An element α^i is a root of $\sigma(x)$ if $\sigma(\alpha^i) = 0$.

This direct substitution method was first utilised by Peterson [50] and later formalised by Chien [51] in the architecture shown in Figure 3.22. The technique is now commonly known as the Chien search. The registers are initially loaded with the coefficients of the locator polynomial:

[8] Two polynomials $f(x)$ and $g(x)$ with coefficients from finite fields are said to be *congruent* modulo m if and only if all coefficients of the difference polynomial $f(x) - g(x)$ are divisible by m.

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2 + \sigma_3 x^3 + \ldots\ldots + \sigma_t x^t \tag{3.34}$$

Multipliers for each power of α are then clocked once, and the results of each $(\sigma_i \alpha^j)$ multiplication summed and compared with zero. If the comparison is true, an error location has been found. The process repeats until all possible roots have been tested.

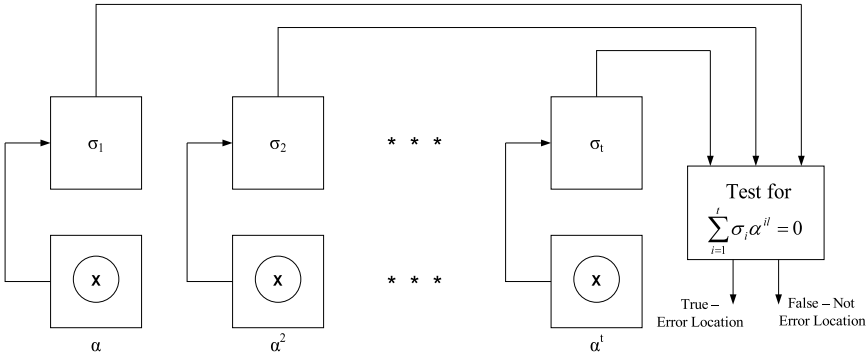

Figure 3.22: *A chien search circuit*

For binary codes, error correction is achieved simply by adding 1, modulo 2 at the appropriate error locations. For the non-binary case (including the Reed-Solomon) codes, an appropriate error value must be determined. Forney [52] has formulated a solution relating the error values y_i to the evaluator polynomial and the derivative of the locator polynomial.

$$y_i = \frac{x^{g_0} w(x)}{x\sigma(x)} \tag{3.35}$$

The derivative of the locator polynomial simplifies to the odd powers of $\sigma(x)$, so the error values may also be readily computed using simple systolic hardware structures. Error correction is finally achieved by adding the values calculated in Equation 3.35 modulo 2 with the received data, and delayed to match the processing time of the engine.

3.5.8 Practical Reed-Solomon Decoding

The steps described above are integral to most practical Reed-Solomon architectures. As has been shown, the Syndrome Calculator is a relatively simple division circuit, much like the encoder structure. Similarly, the Chien search and Forney correction functions lend themselves

naturally to efficient hardware implementation. In contrast, the theoretical difficulties of solving the Key Equation are mirrored in the various existing practical implementations, such that the Key Equation Solver block remains the computational bottleneck in any decoder system. The search for an optimal heuristic KES solution has thus driven significant research effort in this domain.

The literature also reveals some interesting approaches to developing efficient architectures, not exclusively focused on the Key Equation. Shen et al. [53] exploit the similarities in structure between the syndrome calculator and encoder circuit to merge the functions into a single hardware block. Seki et al. [54] have reported improvements in power consumption and logic size based on a time-multiplexed RS decoder structure. At the arithmetic level, Strollo et al. [55] introduce a bit-parallel multiplier structure to improve the efficiency of their CMOS implementation.

As in the case of the basic encoder functions, Aliathon Ltd. had previously developed a suite of efficient arithmetic blocks for Reed-Solomon decoding. Considering this, and in keeping with the thesis statement outlined in 1.3, it was decided that the search for gains and optimisations in FPGA-based Reed-Solomon decoding should focus on possible architectural improvements. Specifically, the goal here was to investigate possible optimisation techniques for OTU-3 systems, for which the current literature again suggested the Key Equation as a promising starting point.

3.5.9 Towards Optimisation of the KES Architecture

As discussed in 3.5.6 the Key Equation Solver (KES) block is typically the most complex in any Reed-Solomon decoder system. Two well documented approaches to solution of the Key Equation dominate the literature. Massey [56] was the first to give a physical interpretation to Berlekamp's mathematics, reducing the problem (like Peterson before) to one of shift-register synthesis. Their resulting heuristic solution has become known as the Berlekamp-Massey algorithm.

This approach and in particular its "inversionless" variants [57, 58] (which remove the need for the computationally intensive inversion of finite field elements), have proven popular in contemporary implementations. In particular, Aliathon's current KES architecture is based on an inversionless Berlekamp-Massey implementation as specified by Sarwate and

Shanbhag [13]. Formal definition of this variant of the algorithm is given in Appendix A.

Equally applicable to solution of the Key Equation is the Extended Euclidean algorithm [36, 59] which applies iterative polynomial division to generate scalar multiples of the error locator and error evaluator polynomials simultaneously. Formal definition of the version presented by Pretzel [36] is given in Appendix B; the algorithm is an intuitive extension of Euclid's division theorem for finding the greatest common divisor of two integers - or polynomials in the context of coding theory.

The Extended Euclidean algorithm has traditionally been favoured in decoder implementations given its relative simplicity (the locator and evaluator polynomials are found simultaneously) which tends to yield very regular, systolic hardware structures and relatively simple control circuitry. In contrast, the Berlekamp-Massey algorithm tends to yield more complex, less regular hardware structures and has been adopted in relatively few implementations, even though (in its inversionless form) it potentially delivers higher performance solutions, which eliminate the need for polynomial division.

Given the relative strengths of each approach, it is perhaps not surprising that more recent research has sought to combine the attributes of the Extended Euclidean and Berlekamp-Massey algorithms. Sarwate and Shanbhag [13] for example, present a reformulation of the inversionless Berlekamp-Massey algorithm to yield Euclidean-like regularity in the resultant hardware structures. Of particular interest in this context is more recent research by Truong et al. [12], which presents an algorithm (hereafter referred to as the "hybrid decoding algorithm") drawing on the mathematical bases of both the Extended Euclidean and inversionless Berlekamp-Massey approaches. The work claims to achieve decoding performance three times faster than that achieved in [58], and was thus deemed a good prospect for further investigation with a view to implementation in an FPGA.

3.5.10 The Hybrid Decoding Algorithm

The hybrid algorithm is similar in structure and initialisation to the Extended Euclidean algorithm, but uses a combination of linear operations and a distributed multiplication based on the Berlekamp-Massey algorithm to remove the need for polynomial division.

3.5.10.1 Formal Definition

Notation:

- $\Omega(x)$: The Evaluator Polynomial
- $\Lambda(x)$: The Locator Polynomial
- δ: A Working field element - commonly referred to as the discrepancy
- γ: A Working field element
- k, l: Integer control variables

For each stage of the algorithm there are three working polynomials; for example in the case of the evaluator in the *k*th iteration, one has $\Omega_k^{(a)}(x)$, $\Omega_k^{(b)}(x)$ and $\Omega_k^{(c)}(x)$.

1. *Initialisation (Based on the Extended Euclidean algorithm).*

Set $\Omega^{(a)}(x) = x^{d_{min}-1}$ where d_{min} is the minimum distance of the code.
Set $\Lambda^{(a)}(x) = 0$, $\Lambda^{(b)}(x) = 1$.
Set $\Omega^{(b)}(x) = s(x)$ where $s(x)$ is the Syndrome Polynomial.
Set $k = 1$ and $l = 0$.

2. *Shift (b) polynomials one position left (equivalent to multiplication by x). Increment k.*

$\Omega_k^{(b)}(x) = \Omega_k^{(b)}(x).x$
$\Lambda_k^{(b)}(x) = \Lambda_k^{(b)}(x).x$
$k = k + 1$ (if not first iteration)

3. *Assign δ and γ from upper coefficients of $\Omega_k^{(b)}(x)$ and $\Omega_k^{(a)}(x)$. Compute current (c) polynomials using multiplication techniques from the Berlekamp-Massey algorithm.*

$\delta = \Omega_{d,k}^{(b)}(x), \gamma = \Omega_{d,k}^{(a)}(x)$
$\Omega_k^{(c)}(x) = \delta.\Omega_k^{(a)}(x) + \gamma.\Omega_k^{(b)}(x)$
$\Lambda_k^{(c)}(x) = \delta.\Lambda_k^{(a)}(x) + \gamma.\Lambda_k^{(b)}(x)$
If $k = d - 1$ then STOP.

4. *Conditionally translate working (a) polynomials for next iteration and update l.*

If $\delta \neq 0$ and $2l = k - 1$ then

Set $l = k - l$

Set $\Omega_{k+1}^{(a)}(x) = \Omega_k^{(b)}(x)$ and $\Lambda_{k+1}^{(a)}(x) = \Lambda_k^{(b)}(x)$

else

Set $\Omega_{k+1}^{(a)}(x) = \Omega_k^{(a)}(x)$ and $\Lambda_{k+1}^{(a)}(x) = \Lambda_k^{(a)}(x)$

end if.

5. *Update working (b) polynomials for next iteration.*

Set $\Omega_{k+1}^{(b)}(x) = \Omega_k^{(c)}(x)$ and $\Lambda_{k+1}^{(b)}(x) = \Lambda_k^{(c)}(x)$

Go To 2.

3.5.10.2 Verification

The first stage in the investigation of the algorithm proposed in [12] was to verify that it was functionally correct. Initially this was proven by hand, using a RS(15,9) codeword which uses 4 bits per symbol and is thus relatively easy to compute manually. The detailed working for an example with a known error vector and syndrome is shown in Figures 3.23 and 3.24.

Note that numbers shown correspond to the power of α of the field element coefficient in any given position. The numbers along the top represent the order of the corresponding position. For example, the final polynomial in the c position (the final entry in the table of Figure 3.24) is given by $\Lambda^{(c)}(x) = \alpha^{13}x^5 + \alpha x^4 + \alpha^9 x^3$. Note also the areas of corresponding shading reflecting where the delta (δ) and gamma (γ) values multiply each coefficient in the *a* and *b* polynomials in Stage 3 of the algorithm. The arrows show where a left shift of the b polynomial has been applied.

The results produced by this example were shown to produce the correct error locations and values. The algorithm was subsequently simulated in VHDL using behavioural models[9] with known correct codewords, producing identical results to those provided by a known-good behavioural Berlekamp-Massey equivalent.

[9] Not intended for logic synthesis.

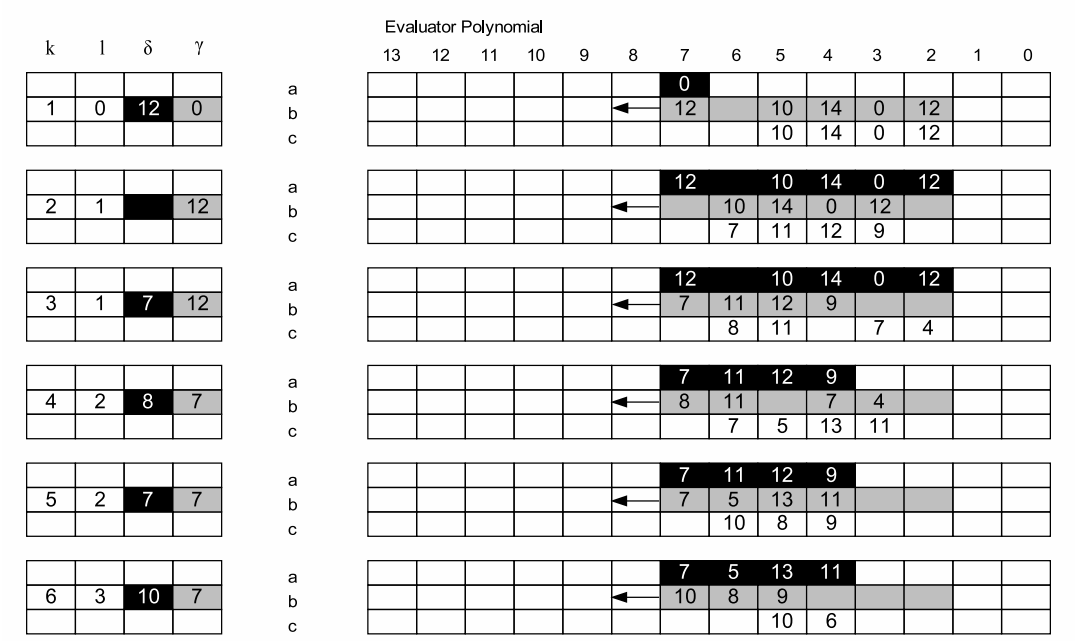

Figure 3.23: *Detailed working of hybrid algorithm for the evaluator polynomial*

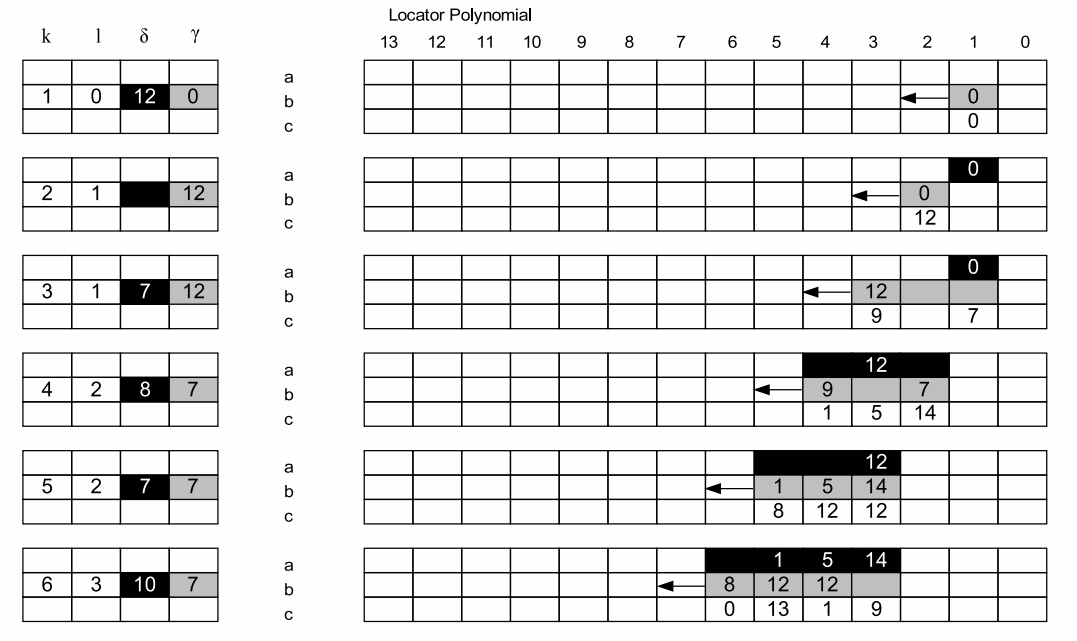

Figure 3.24: *Detailed working of hybrid algorithm for the locator polynomial*

3.5.11 VLSI Implementation

Once a degree of confidence in the basic operation of the hybrid algorithm had been established, the next stage was to consider the practical aspects of any FPGA hardware implementation. A VLSI structure based on a direct implementation of the hybrid decoding algorithm had already been proposed [60, 61] as illustrated in Figures 3.25 and 3.26 for the evaluator and locator computations respectively.

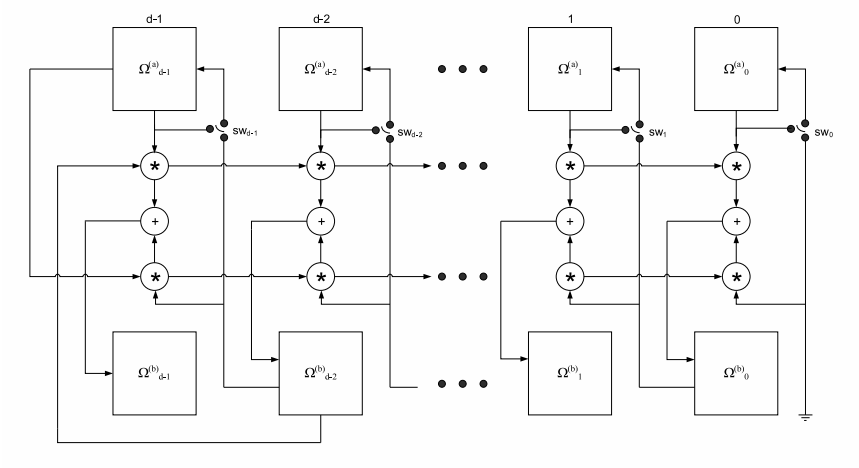

Figure 3.25: *VLSI architecture for evaluator computation based on a hybrid decoding algorithm*

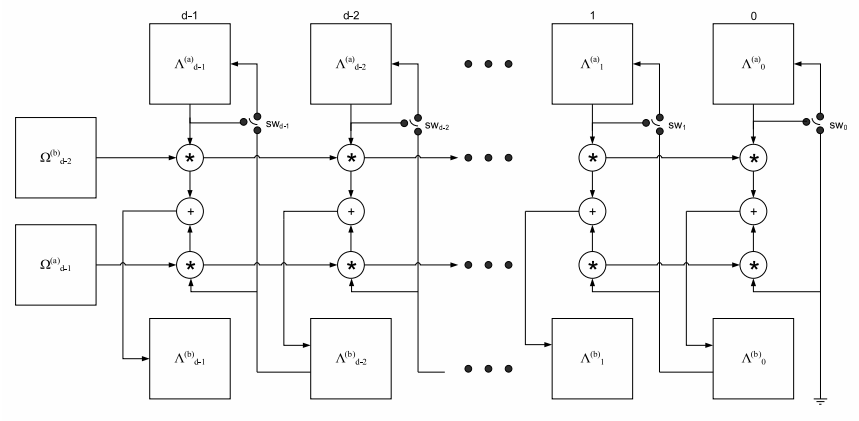

Figure 3.26: *VLSI architecture for locator computation based on a hybrid decoding algorithm*

The architecture comprises banks of registers to store the coefficients of the working polynomials. Cascaded adders and multipliers implement the distributed arithmetic required in Stage 3 of the algorithm, and the update decision for polynomial a required in Stage 4 is

represented in the figure as a series of switches (implemented as multiplexers).

The resultant architecture is highly regular, and thus potentially a good candidate for hardware implementation. However the structure proposed requires $(8t + 4)$ registers, $(8t + 4)$ finite field multipliers and $(4t + 2)$ finite field adders in total to compute the locator and evaluator polynomials, where t is the error correcting capability of the code.

Thus although the direct implementation of the hybrid algorithm completes in only $2t$ clock cycles, initial comparison with Aliathon's existing solution suggested that an FPGA architecture based on the former would be too large to be a credible alternative. With this in mind, further study of the operation of the hybrid algorithm was undertaken, with a view to identifying an improved architecture suitable for implementation in an FPGA.

3.5.12 Towards an improved KES architecture

Whilst a direct implementation of the hybrid algorithm does not yield a particularly efficient solution for OTU-3 systems, investigation of the algorithm's behaviour across the correctable range of error vectors yielded some interesting results pointing the way to an optimised hardware solution.

A synthesizable VHDL implementation of a Key Equation Solver based on the hybrid algorithm was developed, and embedded in a test system comprising known behavioural models of the Syndrome Calculator, Chien Search and Forney Calculation blocks. For comparison, an existing Key Equation Solver (based on the Berlekamp-Massey algorithm and known to be functionally correct), was also embedded in the test code. A behavioural encoder was used to generate valid RS(255,239) codewords. These codewords were corrupted with known error vectors of fixed order from a single error to *t* errors (where again, t was the error correcting capability of the code) and presented as inputs to the decoder.

The behaviour of individual coefficients in the evaluator and locator polynomials was then observed on a per-iteration basis until the algorithm terminated. Coefficients which were non-zero and therefore had to be computed during any given iteration were tabulated as shown in Figures 3.27 to 3.34.

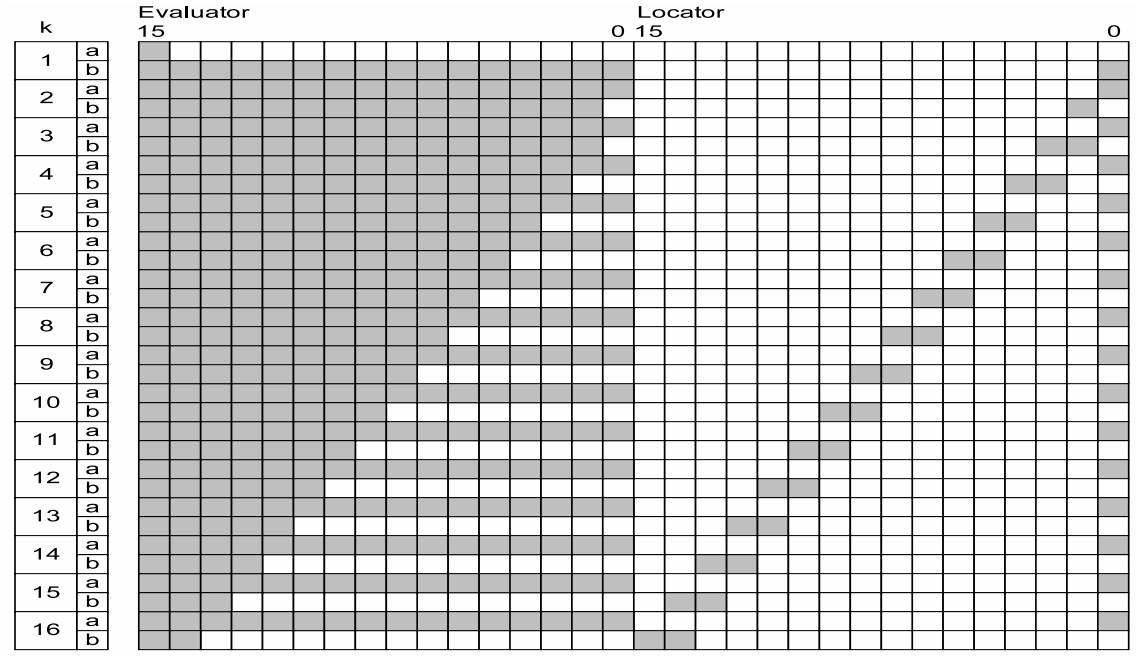

Figure 3.27: *Non-zero coefficients with 1 symbol error*

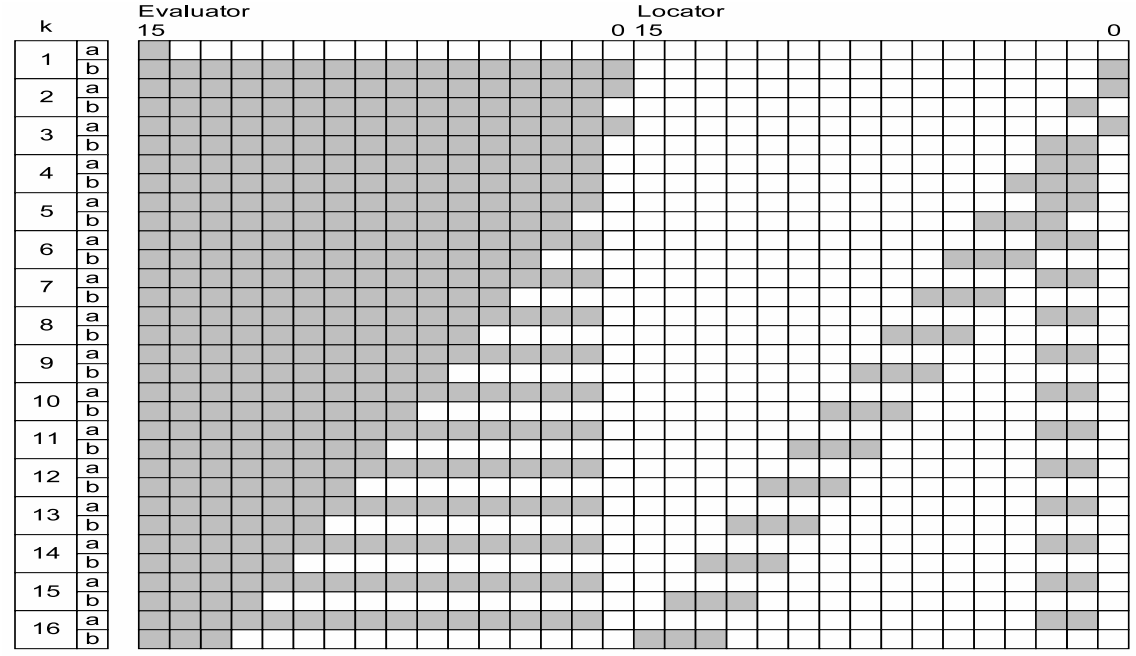

Figure 3.28: *Non-zero coefficients with 2 symbol errors*

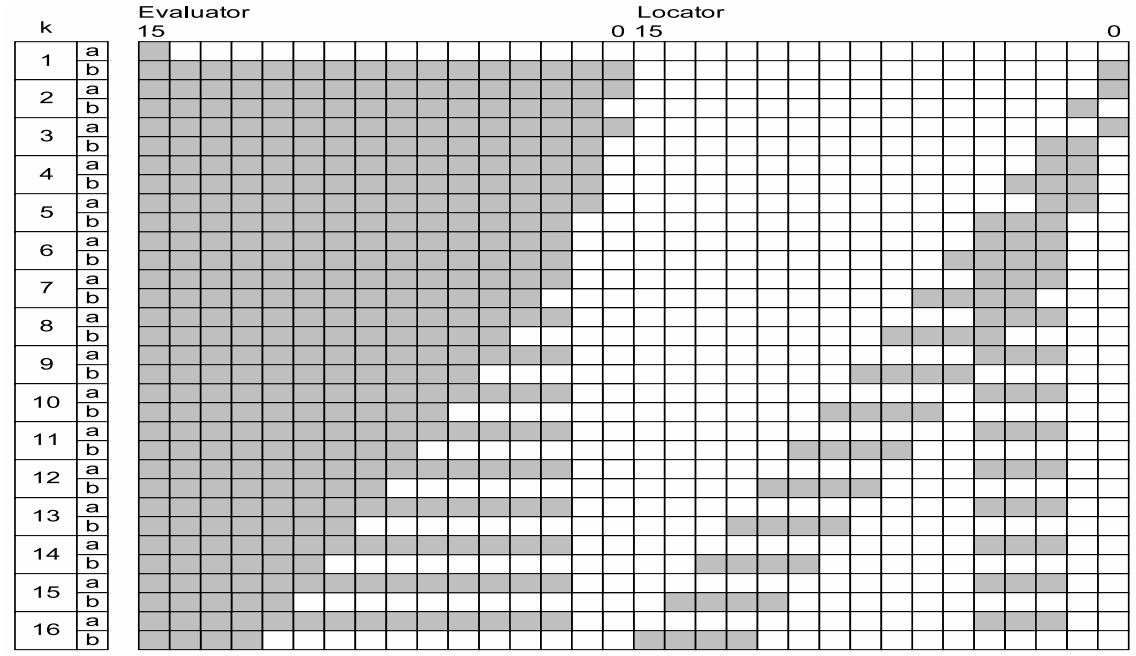

Figure 3.29: *Non-zero coefficients with 3 symbol errors*

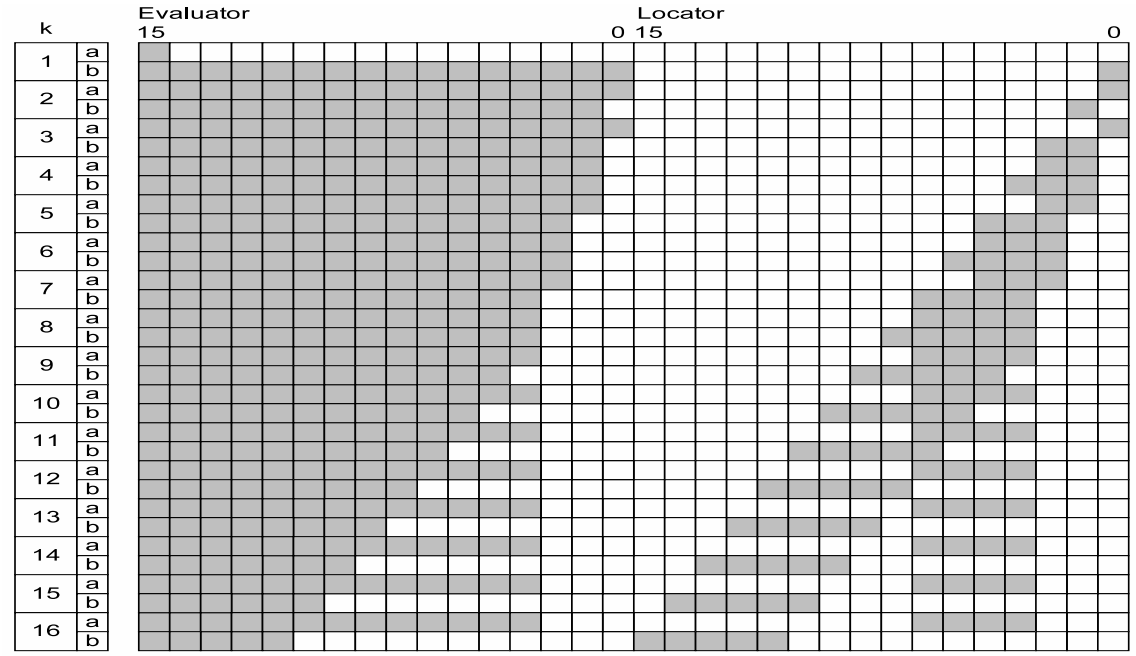

Figure 3.30: *Non-zero coefficients with 4 symbol errors*

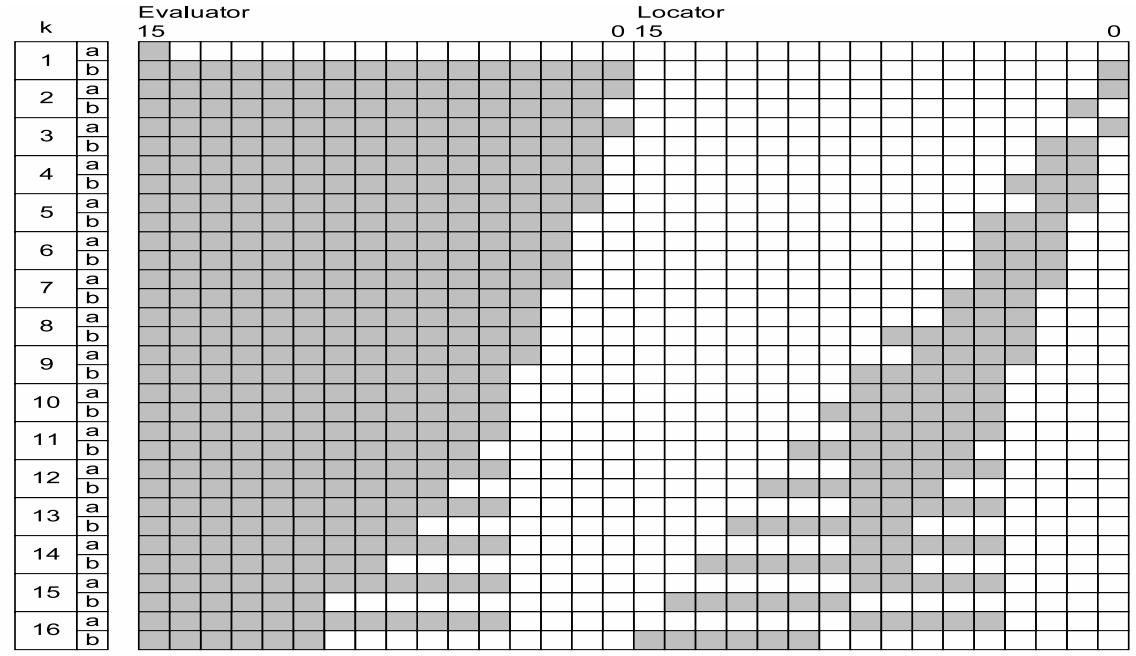

Figure 3.31: *Non-zero coefficients with 5 symbol errors*

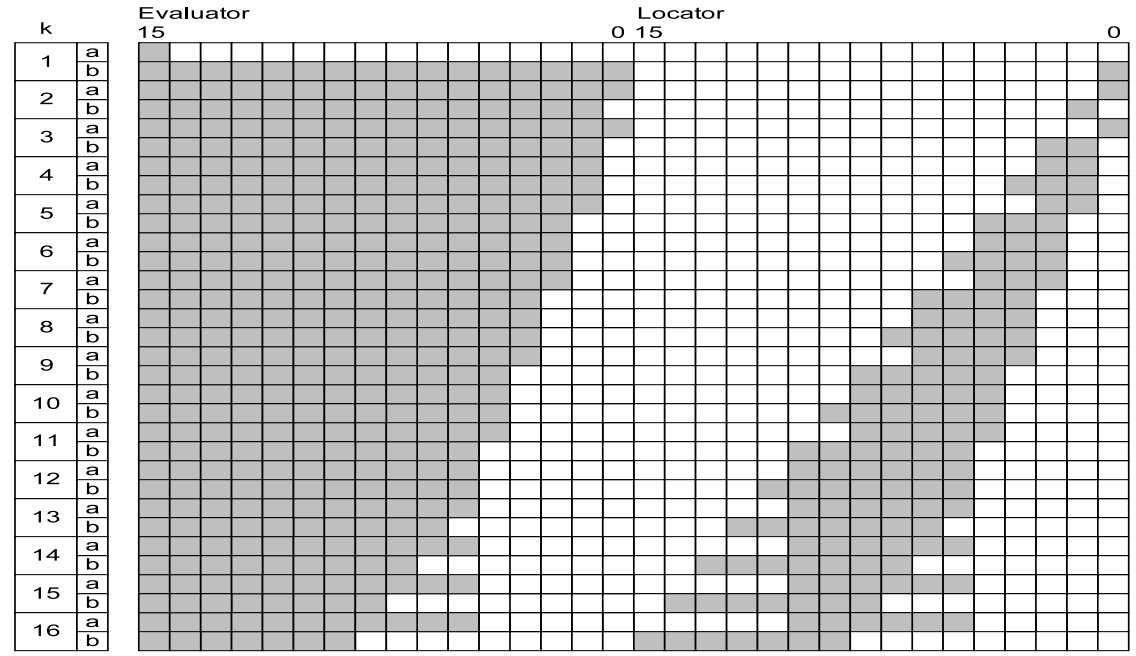

Figure 3.32: *Non-zero coefficients with 6 symbol errors*

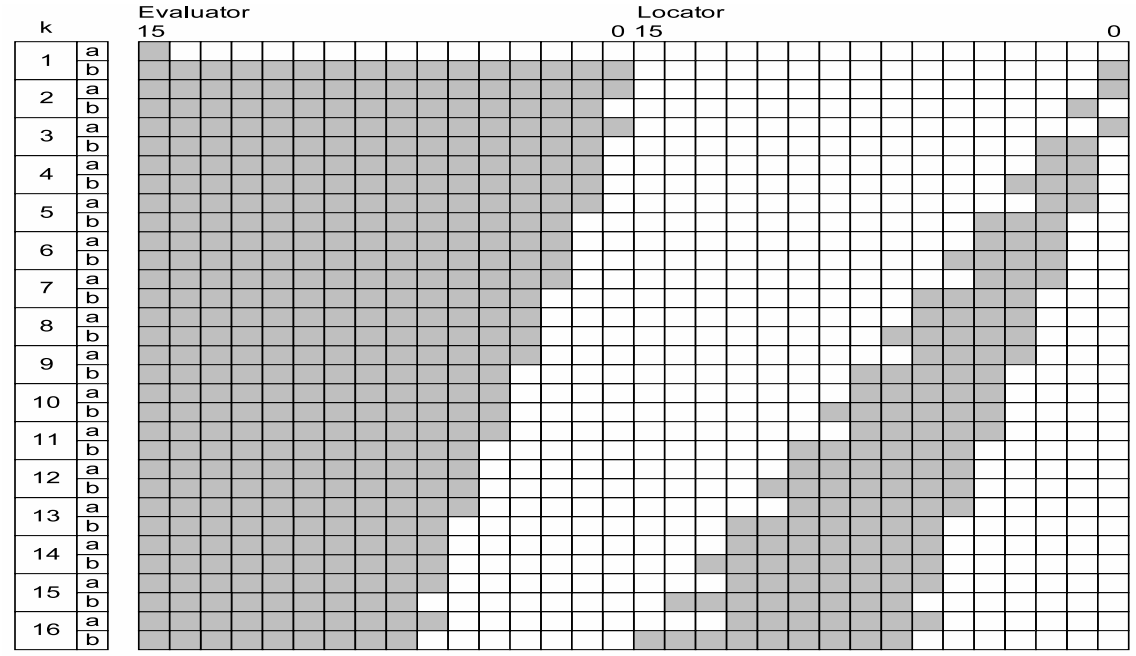

Figure 3.33: *Non-zero coefficients with 7 symbol errors*

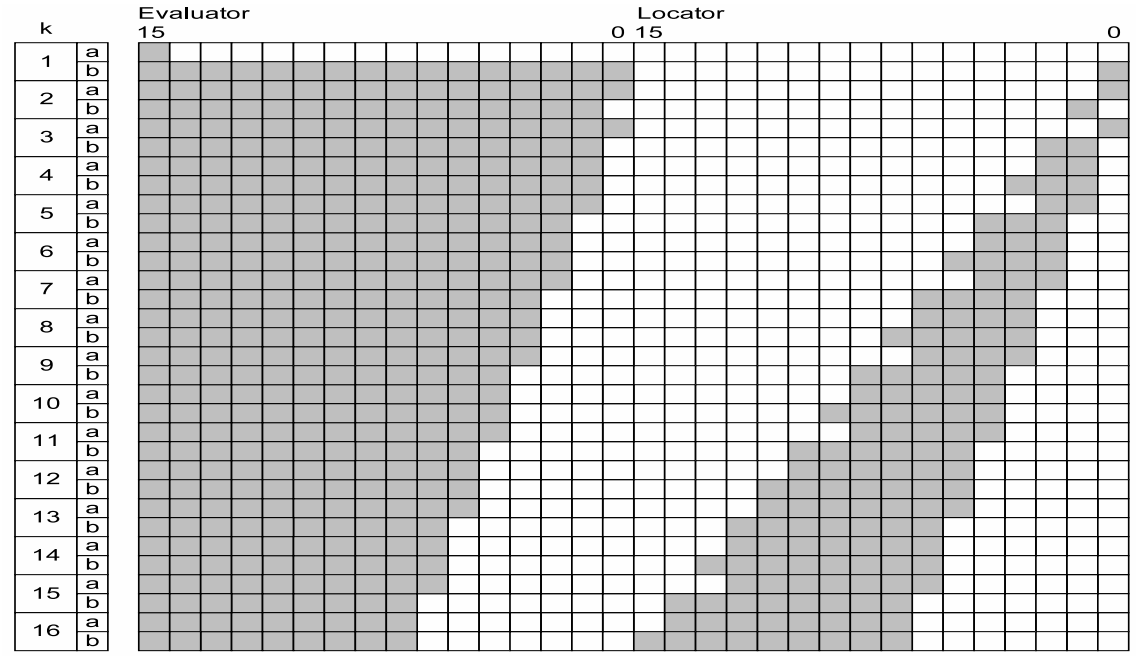

Figure 3.34: *Non-zero coefficients with 8 symbol errors*

From these results, it was observed that as the locator grew in size, coefficients in the evaluator were removed, such that the number of occupied coefficient positions in any given iteration reached a maximum of $(2t+2)$. It was further noted that because the locator polynomial grows in a predictable manner, one may take the opportunity to right-justify its non-zero coefficients, as shown in Figure 3.35 (where the locator coefficients occupy the rightmost portion of the polynomial), such that all of the information required to complete the algorithm may be stored in $(2t+3)$ registers per concatenated polynomial.

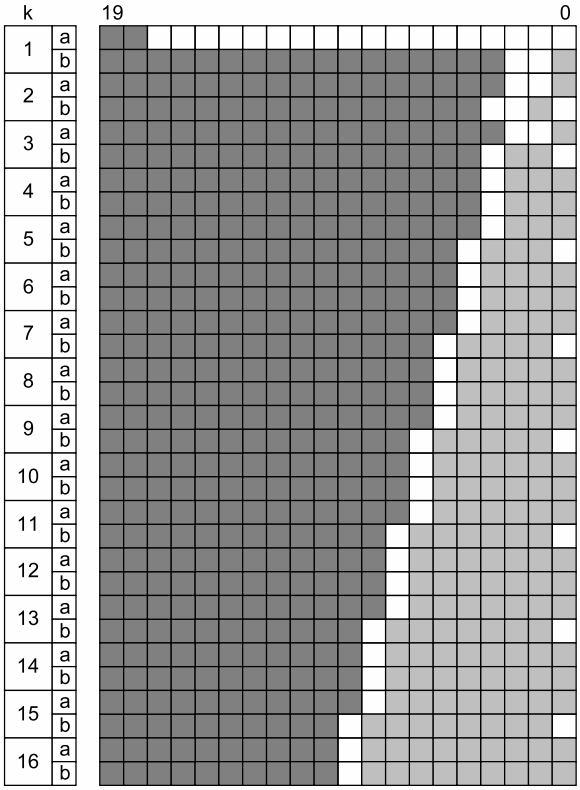

Figure 3.35: *Proposed coefficient justification for worst case error vector*

A further improvement to the existing algorithm can be made, which whilst offering marginal performance improvement in absolute terms, is of key significance to OTU-3 systems. Using the initialization conditions specified in the original definition [12], the algorithm completes in $2t$ iterations, where t is the error correcting capability of the code. However by appropriate transformation of the Syndrome Polynomial inputs and working variables, it is possible to skip the first iteration (since all the information required for the second iteration is presented at initialisation), and allow the algorithm to produce a valid result every $(2t-1)$ clock cycles.

This improvement is integral to the viability of the new decoder for 43Gbps systems. At 43Gbps, one receives 16 Reed-Solomon codewords in 127 clock cycles (processing two symbols per cycle). Using a decoder based on the inversionless Berlekamp-Massey algorithm - previously developed at Aliathon Ltd. - 32 engines are required comprising approximately 350 slices each to give a total utilization of 11200 slices.

For the RS(255,239) 8-error-correcting code, a prototype Key Equation Solver which stores the evaluator and locator as a concatenated polynomial and performs the required polynomial justification has been developed. The architecture decomposes to a series of multiplexers and demultiplexers which control the datapath into the arithmetic blocks associated with the original algorithm. The structure is shown for a single coefficient in Figure 3.36.

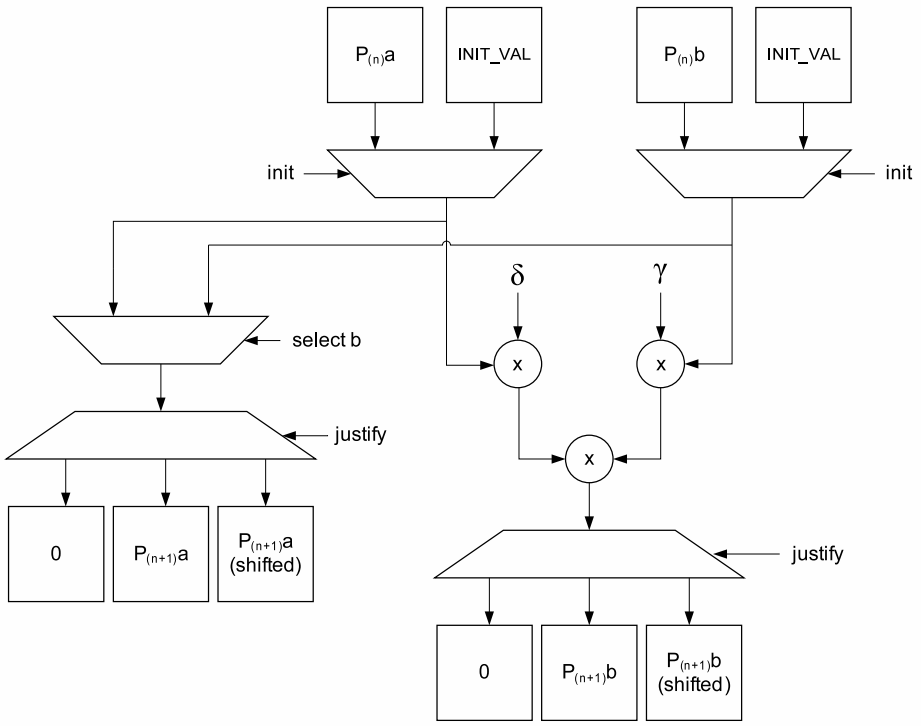

Figure 3.36: *Modified KES architecture for a single coefficient*

Selection logic determines whether the current a coefficient remains unchanged, or whether it is assigned the previous b value, as defined in Stage 4 of the original algorithm. Additional selection logic then determines whether the coefficients of the locator should be justified in the current iteration. The new KES architecture can also exploit the modified initialisation proposed above to complete calculations in $(2t-1)$ clock cycles.

The prototype KES can thus process 1 codeword in 15 clock cycles, or 16 codewords in 240 clock cycles. Therefore with 2 of the new engines, comprising approximately 2000 slices each[10], one may solve the key equation with a total utilization of 4000 slices, or 36% of the original utilisation figure - a significant improvement.

[10] Built for Xilinx Virtex II Pro.

Chapter 4
Packet Classification

4.1 Introduction

Packet Classification is a key enabling function for an increasing number of networking applications, and in fact embodies a diverse range of techniques. In the most general terms, classification applications seek to identify a data packet based on some significant tuple within its header or payload, and associate some action with this identification. Iyer et al. [62] propose the following definition:

> Given a set of rules or policies defining packet attributes or content, **packet classification** is the process of identifying the rule or rules within this set to which a packet conforms or matches.

Within this broad categorisation, packet classification techniques vary in accordance with the numerous application areas. Internet Protocol (IP) routing and switching, Quality of Service (QoS) provision [63], network security and intrusion detection [64], per-flow[1] context and monitoring [65, 66], server load-balancing, filtering and firewalls, and emerging *content aware* applications are all driving a large body of contemporary research in the field.

These diverse application areas create a number of challenges for the system designer. Continued growth in the internet (thought conservatively to be a year-on-year doubling [67]) has brought increased network traffic volume and system line rates, whilst emerging applications sensitive to delay and jitter demand packet forwarding performance significantly beyond traditional *best effort* capabilities.

As an example, a typical OC-192 routing link at 10Gbps receiving minimum-size 40 byte packets would require a classification decision every 32ns [68], based on a rule database typically in the tens of thousands. Similarly, contemporary flow monitoring systems claim 10Gbps throughput supporting 160,000 subscribers and up to 4 million flows [69].

[1] A flow is a group of related packets, typically linked by some addressing commonality.

To more clearly delineate these application areas, and place the contribution of this research in context, it is helpful to distinguish between three key types of Packet Classification; *String Matching*, *Longest Prefix Matching (LPM)*, and *Exact Matching*.

4.1.1 String Matching

The huge increase in internet traffic volume has been mirrored by a commensurate increase in the volume of malicious traffic such as viruses, worms and distributed denial of service (DDoS) attacks, all of which can have a significant effect on network performance [70]. Such exploits typically utilise both the packet header and the payload.

Thus Network Intrusion Detection Systems (NIDS) must have the capability to identify data of interest which may be of variable length, and may occur at any offset within the packet. Furthermore, the payload content may be malicious only in certain contexts, determined by the packet header fields. The problem decomposes to one of String Matching, whereby one seeks to find occurrences of a string or regular expression within another body of text.

Thus NIDS typically comprise some basic packet filtering capability based on the packet header, plus a set of signatures, or strings of interest. Classification depends on establishing a cross-product between these two types of match. For example, based upon identifying a suspect protocol and destination port combination via an exact match in the packet header, one might instruct a string search for an associated malicious signature in the payload.

4.1.2 Longest Prefix Matching

The initial standardisation of the Internet in September 1981 specified that each device connected to an Internet Protocol (IP) network should be identifiable via a unique, 32-bit IP address value. This decision meant that there were 4,294,967,296 unique Internet Protocol version 4 (IPv4) addresses available. In the early days after standardisation this seemed like a virtually unlimited address space, and was subdivided into three principal address classes - Class A (intended for very large organisations with potentially millions of connected hosts), Class B (intended for medium-sized organisations with potentially hundreds of thousands of connected hosts) and Class C (for smaller organisations of a few hundred connected hosts). Addresses within these ranges were assigned to organisations on request, initially without much consideration for how addresses might be efficiently allocated.

By 1993, the inefficiencies in this *classful* addressing scheme started to become significant, creating the likelihood of near-term exhaustion of the Class B network address space, problematic growth in the the size of backbone router tables and the possibility of complete exhaustion of the IPv4 address space in the longer term [71]. Whilst a working group for IP Next Generation (to emerge later as IPv6) would address the latter concern, an interim solution was required to solve the Class B exhaustion and router table issues. This came in the form of Classless Inter-Domain Routing (CIDR) [72].

CIDR eliminated address classes, and supported the concept of aggregation whereby a single router table entry could encompass many individual network addresses as a CIDR *block*. Consider the simple example of Figure 4.1. Four router table entries share the common left-most address bits or *prefix* $130.5.63$ and a common rule to route with high priority. Thus, the entries can be aggregated with a wildcard ($*$) indicating that the router does not care what the right most address bits are (since the same rule is actioned regardless of the contents of these bits).

Four router table entries can thus be replaced with one, assuming one has the capability to identify wildcard entries within IP addresses. CIDR addresses are generally assigned in a way which reflects the topology of the network, thus the *longest prefix* in the routing table which matches an incoming packet's address is generally tied to a forwarding rule representing the most efficient *next hop* for that packet. So, if the incoming packet had destination address 192.168.24.30, and the router forwarding table contained the entries 192.168.* and 192.168.24.*, the latter would be returned as the longest prefix match. LPM techniques are pervasive in Media Access Control (MAC) address-based switching, Asynchronous Transfer Mode (ATM) switching and IP filtering, with matches based on the IPv4 "5-tuple" of source address, destination address, source port, destination port and transport layer protocol.

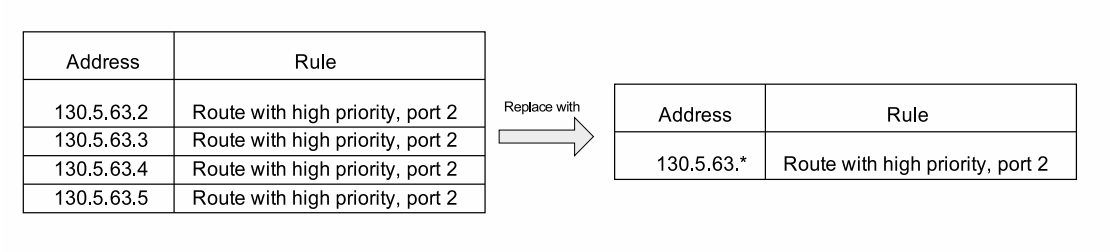

Figure 4.1: *Address grouping with prefix and wildcard*

4.1.3 Exact Matching

Packet routing and forwarding techniques hold an obvious importance for network operation, a fact reflected in the volume of research dedicated to Longest Prefix Match solutions. However, in addition to this fundamental operational level, there are a number of applications including network monitoring, analysis and QoS provision which require more than a prefix-based forwarding decision.

Developed by Cisco Systems, Inc. in 1996 and now the dominant internet monitoring technology, *Netflow* [65] facilitates a wide range of networking applications including accounting and billing, traffic engineering and network security. Packets are assimilated into flow records based on seven unique keys (the IP 5-tuple as discussed in 4.1.2 plus a Type of Service (TOS) byte and Input Logical Interface field (ifIndex)). These keys then drive collation of information on a per-flow basis - how many packets are associated with that flow within a given sampling window? When was the flow created and destroyed? What protocols and service types are being utilised? These questions can be answered only by an exact match on the fields of interest.

There are other applications where an exact match is important. Asynchronous Transfer Mode (ATM) for example, transfers data in 48 byte cells. Thus, to transfer larger packets from other protocols over ATM, the transmitter needs to fragment the data, and the receiver needs to reassemble it in the correct order. This Segmentation and Reassembly (SAR) function is based on an exact match of the Virtual Channel Identifier and Virtual Path Identifier field in the ATM cell label. Similarly, QoS applications such as IntServ and Diffserv implement policies based on exact matching of fields in the packet header. More recent has been the emergence of Pseudo-Wire-Emulation-End-to-End (PWE3) [14] for the provision of guaranteed bandwidth over a packet network. The emulation of traditional wireline services such as E1/T1 over the internet will also be driven by exact match techniques.

4.2 Complexity in Packet Classification

In a generalised sense, the task of packet classification has been shown to have high theoretical complexity [73]. To understand this inherent complexity, it is useful to identify a number of performance metrics which need to be considered in the design of packet classification systems [62, 68, 74].

4.2.1 Space, Time and Power Complexity

Space complexity refers to the upper bound on the space required to represent a rule or filter database. Low storage requirements allow packet classification rules to be stored in lower cost, higher bandwidth commodity static RAM (SRAM) or even embedded SRAM, but as rulesets become bigger and more complex, more expensive higher density, dynamic memory technologies (DRAM) may be required to support them. Practically, one seeks to implement a packet classification scheme in the smallest, lowest cost memory technology available, commensurate with the required line-rate performance.

Time complexity refers to the upper bound on the maximum number of steps or cycles required to make a classification decision. Typically these steps or cycles are memory access times. In many practical scenarios, one seeks to classify at the line rate of the incoming data (up to 40Gbps in emerging core routers). Thus, one seeks to minimise the number of memory accesses required per classification decision. Power complexity refers to the upper bound of the product of the number memory accesses required and the power dissipation cost per memory access.

4.2.2 Update Complexity

Update complexity refers to the upper bound on the maximum number of steps required to perform an atomic insertion or deletion of a rule or filter in the database. In backbone routers and certain firewall applications this may be a less critical metric, where ruleset updates happen much less frequently than normal classification operations. However, in routers with per-flow queuing or network flow monitoring applications, flows may be created and removed much more frequently, such that the dynamic update performance becomes important.

4.2.3 Massive Linearity and Massive Parallelism

The trade-off between space, time and power is ubiquitous in electronic systems design, and unsurprisingly drives much of the literature on packet classification. This research space may be approximately bounded by examining the logical extremes of the performance metrics just discussed.

Consider a classification database of n entries. Any existing entry may or may not match the current entry being processed. Such a match could be established using an exhaustive search

where one must compare the current entry against every existing entry in the classification database. One could compare the current entry with each entry in the database sequentially, accessing one database entry per memory access time. This massively linear search has space complexity $O(n)$ (requiring nominally one location in memory space for each entry in the database) and time complexity $O(n)$ (requiring one memory access time for each entry in the database and a search through the entire database in the worst case).

Alternatively, one could compare the current entry with the entire database in a single memory access time. This massively parallel search has space complexity $O(n)$, but time complexity $O(1)$. This is in fact the approach taken in Content Addressable Memories (CAMs), which dominated early lookup systems, and latterly Ternary Content Addressable Memories (TCAMs) which added the ability to store wildcard bits required for massively parallel Longest Prefix Matching. In comparing linear and parallel approaches to exhaustive search it is also important to note that computational complexity scales linearly with parallelism. So although the linear and parallel searches have the same nominal space complexity $O(n)$, the latter is far more computationally expensive.

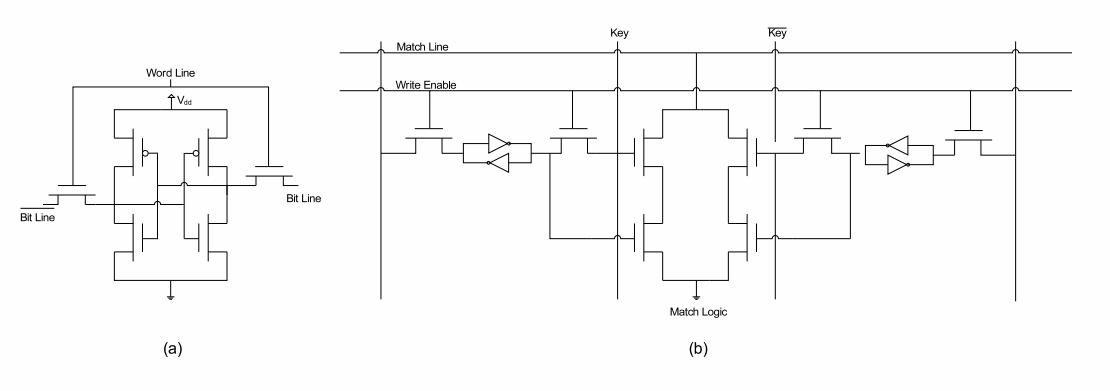

Figure 4.2: *Standard SRAM(a) and TCAM(b) cells*

4.2.4 Towards the Middle Ground

In reality, both massively linear and massively parallel approaches to packet classification are impractical. As core and edge router tables expand to accomodate hundreds of thousands of entries and more complex policies, and flow monitoring requirements extend to potentially millions of subscribers, the time taken to linearly search through every possible match candidate

becomes prohibitive at all but the lowest line rates. (T)CAMs are not a panacea either, suffering from high cost per bit compared to standard memory technology (standard TCAM cells are almost 3 times bigger than an equivalent SRAM cell [75]), storage inefficiency, high power consumption, limited scalability to long input keys, and limited density [76]. Given the limitations just outlined, it is unsurprising that much of the published work on packet classification in fact occupies the algorithmic middle ground between a naïve linear search on one hand and a brute-force architectural solution on the other. This is the focus of the research which follows.

Of the three techniques discussed - string match, longest prefix match and exact match - the last has been identified as of particular commercial interest to Aliathon Ltd. The development of an FPGA-based exact match technology offers balanced risk and technical progression for the company, facilitating the immediate enhancement of legacy products for ATM, and simultaneously creating a platform on which to build a suite of packet-processing IP cores in the future. This commercial emphasis is reflected in the following discussion. A general overview of contemporary string matching techniques is followed by a discussion (in a little more detail) of current longest prefix match techniques. Finally, the state-of-the-art in exact-match classification is discussed, as the principal background material for the current project.

4.3 Techniques for String Matching

As already discussed in 4.1.1 the ability to search through packets and identify potentially malicious content is integral to Network Intrusion Detection Systems, and reported to account for 70% of total execution time in Snort [77]. Classical string matching techniques originally pioneered for text-based search functions (the UNIX *grep* command for example) are thus gaining renewed significance as the basis for compact and fast string matching techniques.

4.3.1 Aho-Corasick String Matching

Based on the Knutt-Morris-Pratt algorithm [78], the seminal Aho-Corasick technique [79] provides algorithms for the construction of an efficient finite state pattern matching machine. The authors summarize the problem as follows: A *string* is a finite sequence of symbols. If $K = y_1, y_2, ..., y_k$ is a finite set of strings called *keywords* and x is an arbitrary text string, one seeks all substrings of x which are keywords in K, even if these substrings overlap.

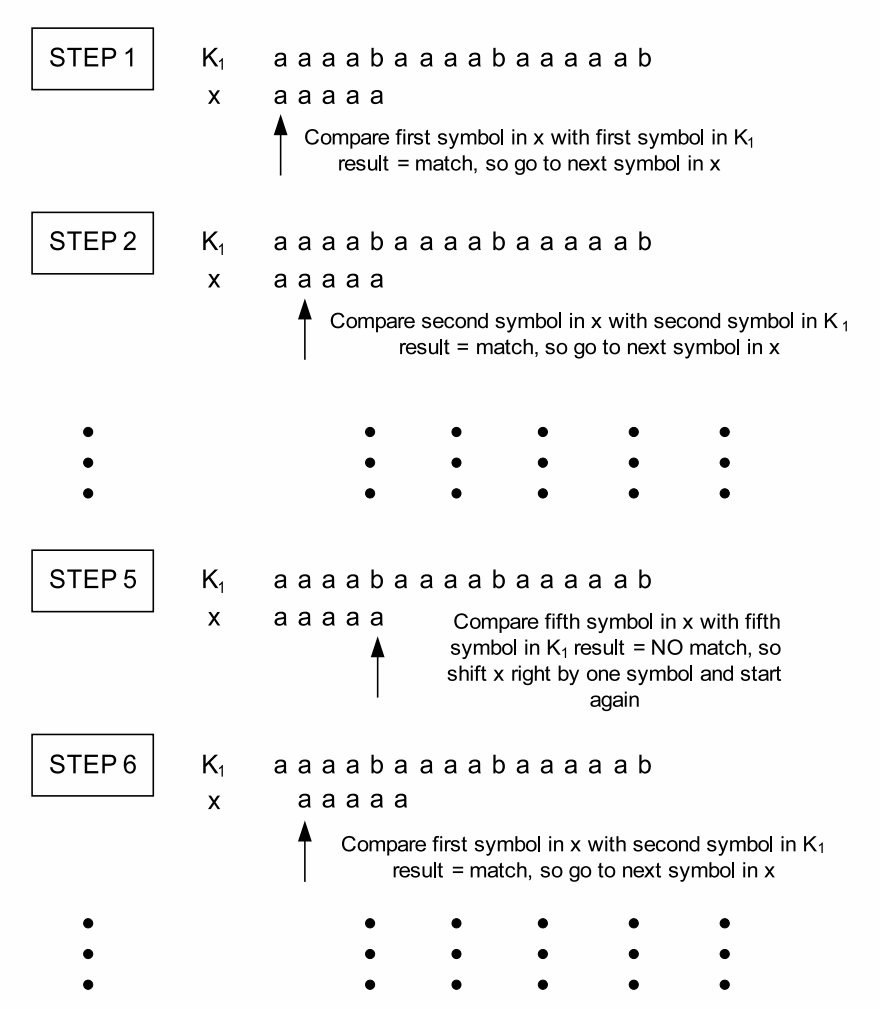

Figure 4.3: *A naïve string matching algorithm*

As a simplified initial illustration, consider a naïve algorithm to search for a single text string *"aaaaa"* within a larger portion of text *"aaaabaaaabaaaaab"* as shown in Figure 4.3. After mutually left aligning the two strings, each symbol in x is compared with its aligned symbol in K_1 in turn until a mismatch is found, whereupon x is shifted right by one symbol and the symbol-wise comparisons begin again.

This shifting mechanism is where the Knutt-Morris-Pratt/Aho-Corasick approaches dramatically improve over the naïve implementation. Observing the repetition of the "a" symbol in x and K_1 it may be noted that there is no chance that a match will be returned by STEP 6 (or indeed by the next three shifts in x). The first mismatch thus results in a multi-symbol shift as shown in Figure 4.4. This optimised pattern shifting based on

pre-processing the strings of interest is designed into the *next-state* logic in Aho-Corasick state machines.

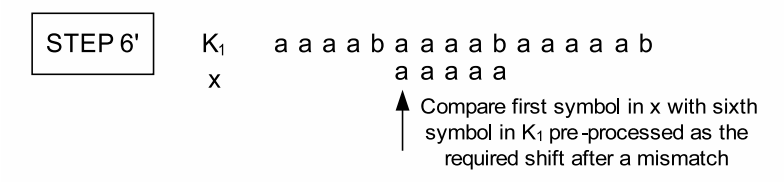

Figure 4.4: *Pre-processed optimised pattern shift*

4.3.2 Boyer-Moore String Matching

Better still is the Boyer-Moore algorithm [80], which does not rely on repetition in the search string for performance improvement over a naïve search. Figure 4.5 illustrates a Boyer-Moore search in another simple example[2], searching for the string "pill" within a longer string "the caterpillar".

Boyer-Moore analyses the search string x from right to left, whilst shifting it (again on the basis of the symbol match result) from left to right along K_1. Optimal shifts are based on a pre-processed array containing an indication, for every possible symbol, if and where it exists in the search string, and thus how far the string can be shifted in the event of a mismatch. More generally, if the length of x is M, one starts by comparing the *last* symbol in x with the Mth symbol of K_1,K_{1_m} . If there is a mismatch, one looks for the *rightmost* occurrence of K_{1_m} in x and shifts accordingly. If K_{1_m} does not occur in the search string at all, then one can safely shift it right by M symbols before the next comparison.

If one assumes that on average the majority of possible characters do not appear in the search string, then the number of character comparisons required approaches N/M, where N is again the length of K_1 and M is the length of x; an improvement over The Knuth-Morris-Pratt algorithm which requires N comparisons.

4.3.3 Contemporary String Matching Solutions

Some recent work in string matching builds on the classical techniques discussed in 4.3.1 and 4.3.2. Wu and Manber [81] note the difficulty in extending the Boyer-Moore approach to the

[2]From www.cee.hw.ac.uk/~alison/ds98/node78.html ©Alison Cawsey.

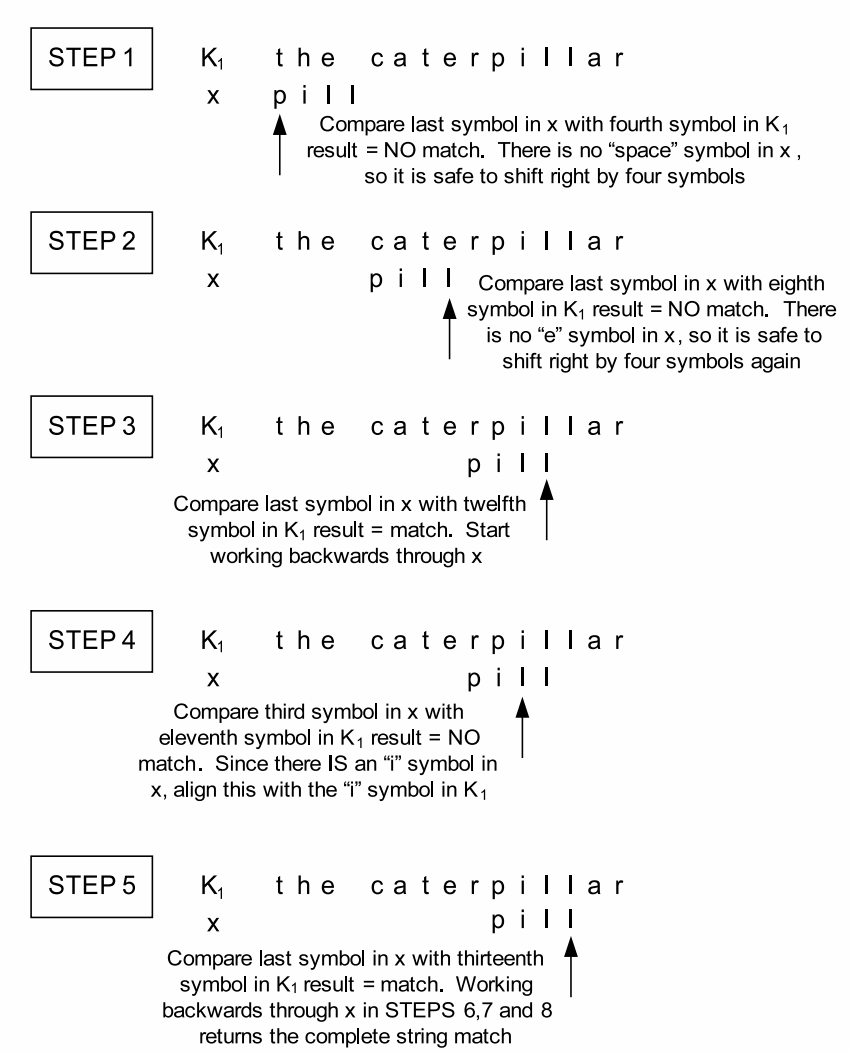

Figure 4.5: *Boyer-Moore pattern shifting*

multi-string matching problem, where (as in Network Intrusion Detection Systems) one seeks to match potentially tens of thousands of patterns. They note that when there are many strings to match, is becomes likely that most of the possible symbols will match in at least one of the search strings, reducing the number of multi-symbol shifts possible and degrading performance.

They propose the imposition of a minimum pattern length, and a novel string pre-processing stage to preserve the speed of the Boyer-Moore approach for multi-string matching. The Aho-Corasick approach has also been subject to more recent optimisation, notably by Tuck et al. [82], who apply the ideas of node and path compression (in the spirit of [83] and [84], discussed in 4.4.3) to dramatically reduce the size of the hardware implementation to 2% of

the original. Clark and Schimmel [85] also leverage the use of finite state machines in their design of multi-character decoders optimised for FPGA logic, claiming 10Gbps performance for current data-set sizes.

4.4 Techniques for Longest Prefix Matching

Taylor [76] has already presented a good taxonomy of packet classification techniques applicable to Longest Prefix Matching. An exhaustive review is not repeated here, since longest prefix techniques are not the focus of this project. Nonetheless, longest prefix matching dominates the literature, and no discussion of packet classification would be complete without some reference to this large body of work.

Contemporary techniques in this domain owe much to Gupta and McKeown who, having identified that the classification problem was unsolvable in the worst case [86], suggested the use of heuristics based on the inherent structure of routing databases.

4.4.1 Recursive Flow Classification

Leveraging and improving on techniques based on a cross product of packet fields [87], a seminal contribution in the application of heuristics to the packet classification problem is Gupta and McKeown's Recursive Flow Classification [88]. The technique is based on exploiting the empirically determined structure in any given dataset. The authors' analyses of real datasets returned a number of interesting characteristics, namely:

- That typically a maximum of 8 fields were specified in classifier rules. Source/destination address, source/destination transport-layer port numbers, type-of-service field, protocol field and transport layer protocol flags.
- That 17% of all rules had only 1 field specified, 23% had 3 fields specified and 60%had 4 fields specified.
- That the transport-layer protocol field is restricted to a small set of values.
- That it is common for many different rules in the same classifier to share a number of field specifications.

Network Layer Destination (32 bits)	Network Layer Source (32 bits)	Transport Layer Destination (16 bits)	Transport Layer Protocol (8 bits)
152.163.190.69	152.163.80.1	*	*
152.168.3.0	152.163.200.157	eq www	udp
152.168.3.0	152.163.200.157	range 20-21	udp
152.168.3.0	152.163.200.157	eq www	tcp
152.163.198.4	152.163.160.0	gt 1023	tcp
152.163.198.4	152.163.36.0	gt 1023	tcp

Table 4.1: *A simple classifier dataset*

The final point is illustrated by the simple classifier dataset of Table 4.1, taken directly from [88], and is the key observation in establishing a heuristic for reducing classification complexity. The authors view the classification task as one of reduction (or decomposition). For example, in the case of the transport-layer destination in Table 4.1, they note that although the field is nominally specified over 16 bits, there is considerable structure (or repetition) within the field. In fact, the possible values can be represented as only four sets $\{$www=80$\}$, $\{$20/21$\}$, $\{> 1023\}$ or $\{$all remaining numbers in the range 0-65535$\}$. Allocating a unique code (or equivalence class identifier - eqID) to each of these sets requires just two bits, hence the reduction in complexity.

These equivalence class identifiers form the basis of the lookup, illustrated in Figure 4.6. For the simple dataset of Table 4.1, each incoming packet is split up into its constituent fields. The destination address, source address, transport-layer port and transport-layer protocol are used directly as indices into four parallel memories.

The contents of these memories have been pre-processed to return the correct equivalence class identifier. There are three unique destination addresses; 152.163.190.69, 152.168.3.0 and 152.13.198.4. Thus the eqIDs can be encoded using just two bits. By inspection, it can be seen that two-bit eqIDs are sufficient for each of the other fields in the first phase lookup. Similarly, in the second phase lookup it can be seen that there are four unique combinations of source and

destination address encoded using three bits, and four unique combinations of transport-layer port and transport-layer protocol, encoded using three bits.

In this case, eqIDs from the first-phase destination and source address lookup, and eqIDs from the first-phase transport-layer port and transport-layer protocol lookup are concatenated and used as indices into the second phase memory. The eqID returned from this second phase lookup is the unique six-bit identifier for the current packet.

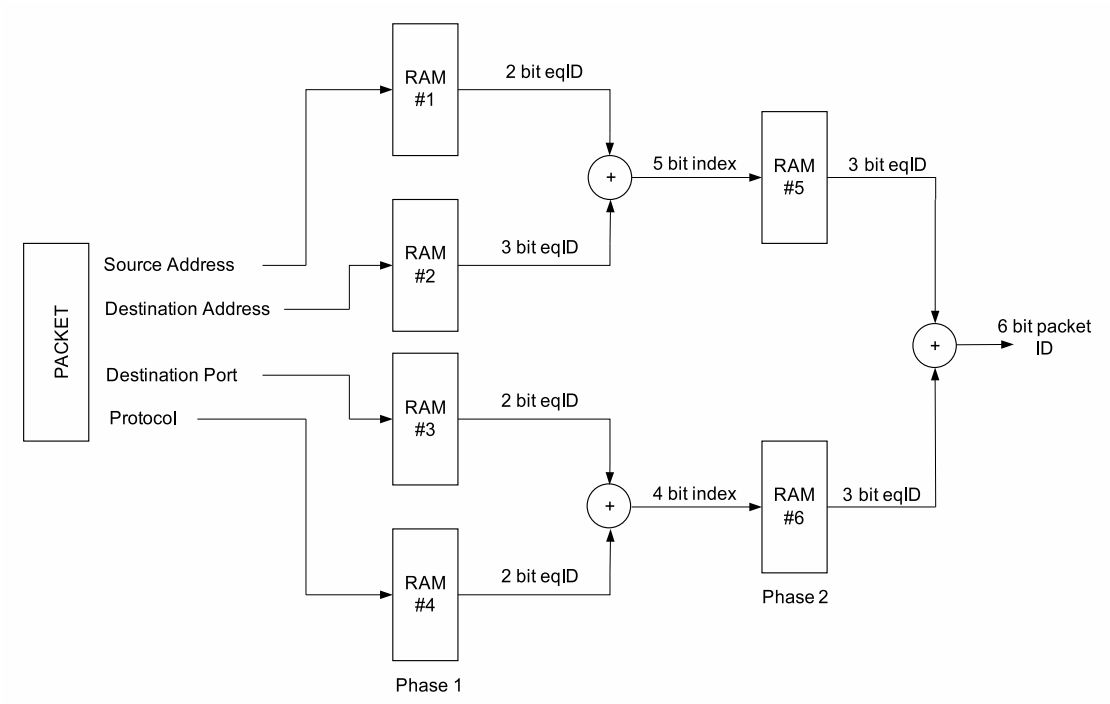

Figure 4.6: *Example lookup using recursive flow classification*

Recursive Flow Classification offers good throughput performance - approximately 30 million packets per second with a 125MHz system clock in pipelined hardware - but at the cost of memory inefficiency; the solution does not scale well to large classifiers. Additionally, the pre-processing requirements make system update difficult when these updates happen frequently.

Filter	Network Layer Destination Prefix	Network Layer Source Prefix
F_1	0*	10*
F_2	0*	01*
F_3	0*	1*
F_4	00*	1*
F_5	00*	11*
F_6	10*	1*
F_7	*	00*
F_8	0*	10*
F_9	0*	1*
F_{10}	0*	10*
F_{11}	111*	000*

Table 4.2: *A simple classifier dataset specified over source and destination address only*

4.4.2 Grid of Tries

A popular approach to classification is the construction of a decision tree, or trie[3]. The incoming packet is split into bits or groups of bits, and these are used to make branching decisions in the trie structure. The rules or filters are typically stored in the leaf nodes of the structure. Again, following [76] consider the simple filter set specified over source and destination addresses in Table 4.2.

Shown in Figure 4.7(a), the top half of the simple Set Pruning structure as defined in [89] represents all the possible destination address prefixes in the classifier, with pointers into the source address space from each node where a prefix match in the destination address space occurs. For example, consider a bitwise search for the destination/source pair $00* / 11*$. One makes two *zero* (left-hand) transitions from the root node, where a pointer (P1) into the appropriate source address sub-trie is found. One then makes two *one* (right-hand) transitions until a leaf node which contains filter F5 is reached; the best match for the destination/source pair.

Whilst such a structure is an elegant visualisation of bitwise classification, it is inefficient, with filters duplicated at multiple leaf nodes. To improve efficiency Srinivasan et al. [87] proposed

[3] From information re*trie*val.

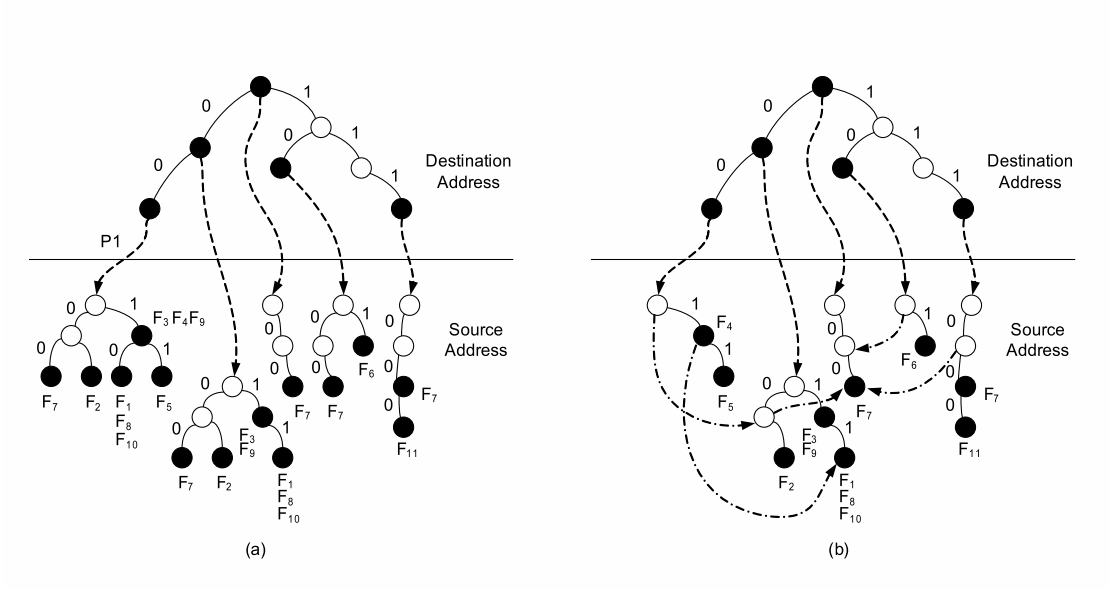

Figure 4.7: *Set Pruning (a) and Grid of Tries (b) structure for the classifier of Table 4.2*

the Grid of Tries, Shown in Figure 4.7(b) where only one instance of each filter is stored in the structure, with searches directed by switch pointers to potentially matching filters. This results in reasonable memory efficiency and search time; the authors quote 2MB memory utilisation to implement a classifier of 20,000 IPv4 filters (considering source and destination prefix fields only).

A weakness of the Grid of Tries approach is poor scalability to classifiers beyond two dimensions, where replication of data structures becomes necessary. However after studying the characteristics of core router classifiers used by Tier 1 Internet Service Providers, Baboescu et al. [68] argue that two dimensional classification remains powerful. They extend the heuristics of Recursive Flow Classification by noting that packets typically match at most a few distinct source-destination prefix pairs present in any ruleset. Thus, in their Extended Grid of Tries scheme, even if the classifier is large, after pre-filtering on source-destination prefix, one is left with a set of remaining filters which is small enough to be searched linearly.

Scalability to multiple field classification is addressed in another seminal scheme proposed by Lakshman and Stidialis [73]. Considering again the destination and source prefixes shown in Table 4.2. In the Bit Vector scheme, two tries are constructed, one for the destination prefixes and one for the source prefixes as shown in Figure 4.8.

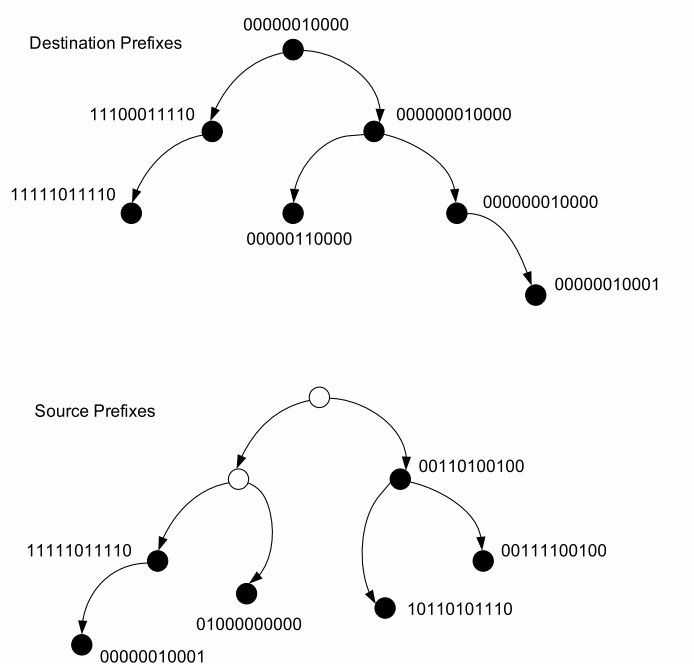

Figure 4.8: *Bit Vector structure for the classifier of Table 4.2*

Each node in the structure is labelled with an N-bit vector, where N is the number of filters in the classifier. Bit j in this vector is set if the corresponding filter in the classifier matches the prefix corresponding to the node. For example, in the destination prefix trie the left-most node corresponding to prefix $00*$ is labelled 11111011110 since it could match filters $F_1, F_2, F_3, F_4, F_5, F_7, F_8, F_9$ and F_{10}. When a multi-field packet header arrives in the Bit Vector scheme, the trie for each field is traversed to return the bit vector label corresponding to any match in that trie. A multi-field match is then returned based on the bitwise intersection of these bit vectors.

Baboescu and Varghese [90] improve on this scheme, again by applying a heuristic approach to the base algorithm. They note that in the databases they investigated (of the order of 100,000 filters) that packet headers typically matched at most four filters. As a result, they conclude that vectors in the original bit vector scheme are typically very sparsely populated with set bits, and reading 100,000 bits per node is inefficient. They propose aggregating *A* bits of the original bit vector into a single bit (which represents the *bitwise or* of the aggregated bits) to reduce the required memory accesses.

4.4.3 Expanded Tries - From Controlled Prefix Expansions to Tree-Bitmap

Some elegant extensions to the *unibit* structure of Figure 4.7 have been proposed, in the form of expanded or *multibit* tries, which seek to process multiple bits at a time to reduce the number of memory accesses required per classification. Consider another simplified set of prefixes in Figure 4.9.

Prefix Name	Prefix
P1	*
P2	1*
P3	00*
P4	101*
P5	111*
P6	1000*
P7	11101*
P8	111001*
P9	1000011*

Figure 4.9: *Simple prefix database*

To process these prefixes say, three bits at a time, one needs to deal with prefix lengths which do not fit naturally into multiples of three bits. Srinivasan and Varghese [91] have proposed the technique of Controlled Prefix Expansion to accommodate precisely this scenario. The idea is to extend a prefix like $1*$ (P1) into all its possible three bit expansions - $100, 101, 110$ and 111. In this case, the expansions 101 and 111 collide with prefixes P4 and P5 respectively. To account for this P1 is given lower priority, since P4 and P5 represent the longest prefix match. Expansion prefixes which collide with existing longer prefixes are in fact redundant, and may be discarded.

The nodes of a three-bit expanded trie with Controlled Prefix Expansion for the simple prefixes of Figure 4.9 are shown in Figure 4.10 [84]. Each node element has two entries. One entry holds a rule or filter matching the prefix (simplified to the prefix name in the figure) and the other is reserved for a pointer to a child node in the trie. The search terminates when a null pointer is reached.

For example, consider searching for an input 111010. The first three bits 111 are used as an index into the root node of the trie, where prefix P5 is stored. Since the associated pointer

is not null, P5 is not the longest prefix match and the search continues, following the pointer into the right-most child node. The second three bits in the search key 010 are used as the index into this node, where prefix P7 is stored. Since the associated pointer is null, the search terminates, returning P7 as the longest prefix match. The authors quote storage requirements after optimisation for the North American Mae-East exchange database at around 500KBytes.

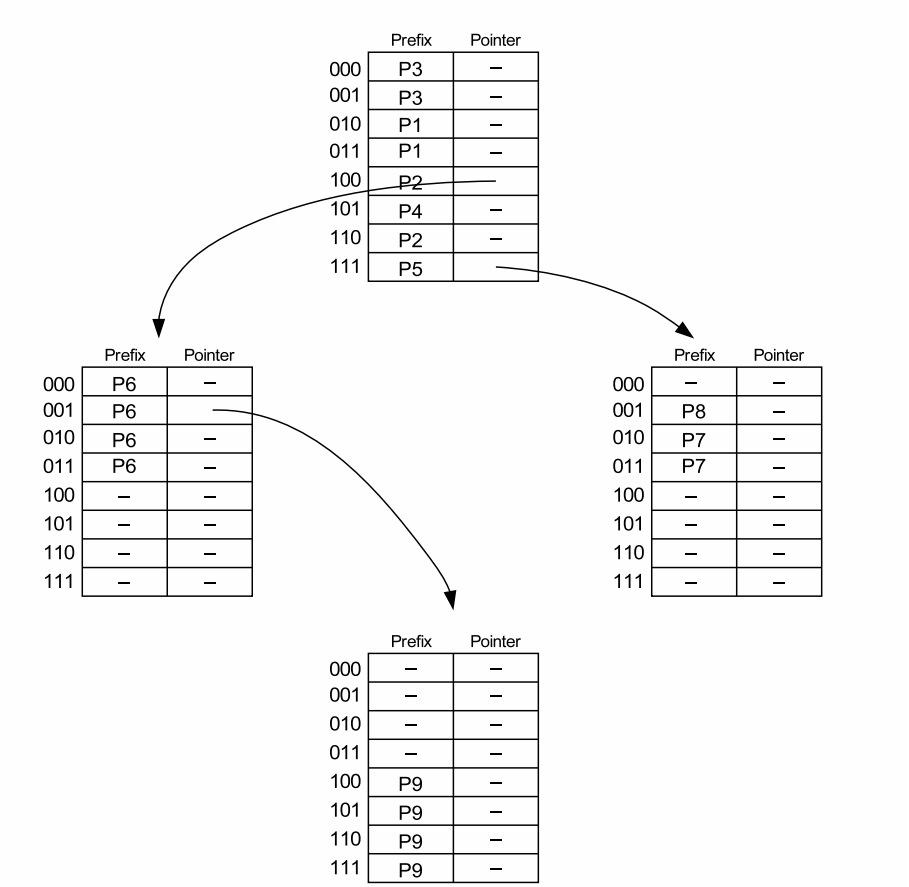

Figure 4.10: *A multi-bit trie with Controlled Prefix Expansion*

The Lulea scheme [83] improves on the basic multibit trie structure by reducing the amount of data stored at each node using a technique called Leaf Pushing. Rather than storing *both* a rule and a pointer at each index in the node, only one of these are stored in the Lulea Trie. In the event that a filter and a valid pointer coincide, the filter is *pushed* into all the free locations in the downstream node indicated by the pointer. The search terminates when a valid filter is found.

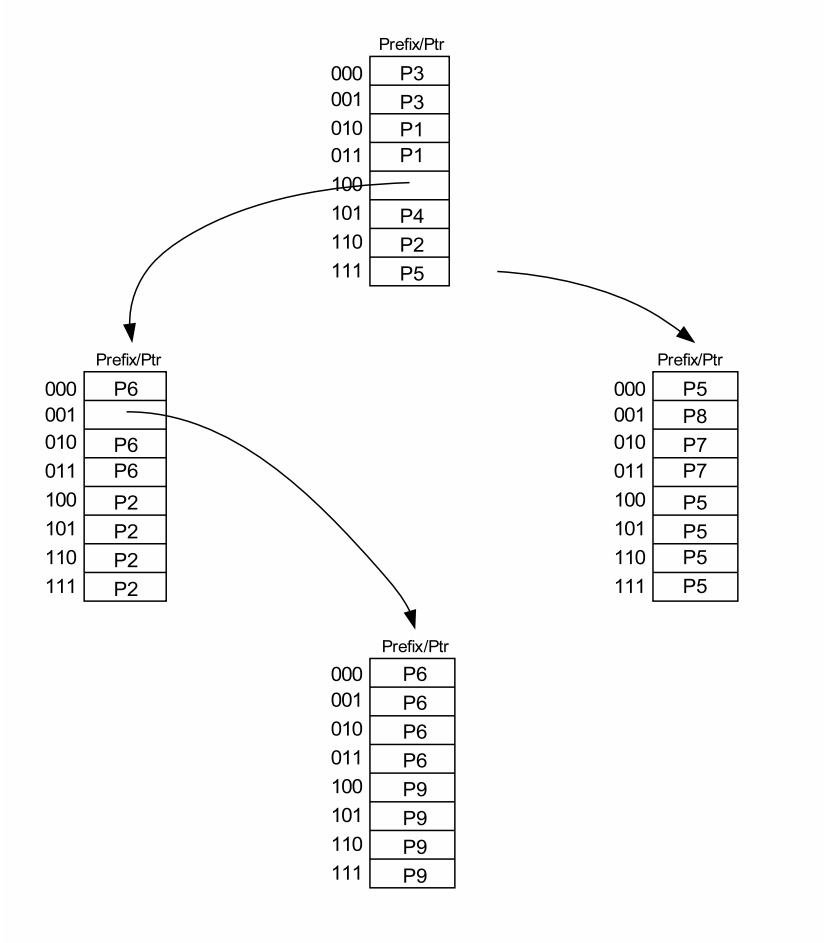

Figure 4.11: *A multi-bit trie with Controlled Prefix Expansion and Leaf Pushing*

Storage efficiency is improved further in the Lulea scheme by defining a bit vector at each node which indicates where filters have been duplicated and thus removes the need to store those filters explicitly. Consider again the structure in Figure 4.11. The nodes can be compressed as shown in Figure 4.12. The first instance of given filter is stored explicitly, and its presence indicated by setting the corresponding bit in the vector. Any *consecutive* duplicates which follow are not stored explicitly, but indicated by setting the corresponding bit in the vector to zero. The bit vector thus becomes a signature for the filters stored at any given node.

Decoding according to the Lulea scheme is best illustrated with a simple example. Consider searching for an input 111111. Since (in this case) three bits at a time are being processed, the first three bits, 111 are used as an index into bit seven of the bit vector in the root node of the trie. Then, starting from the left most bit in the vector, the number of bits equal to one, up to

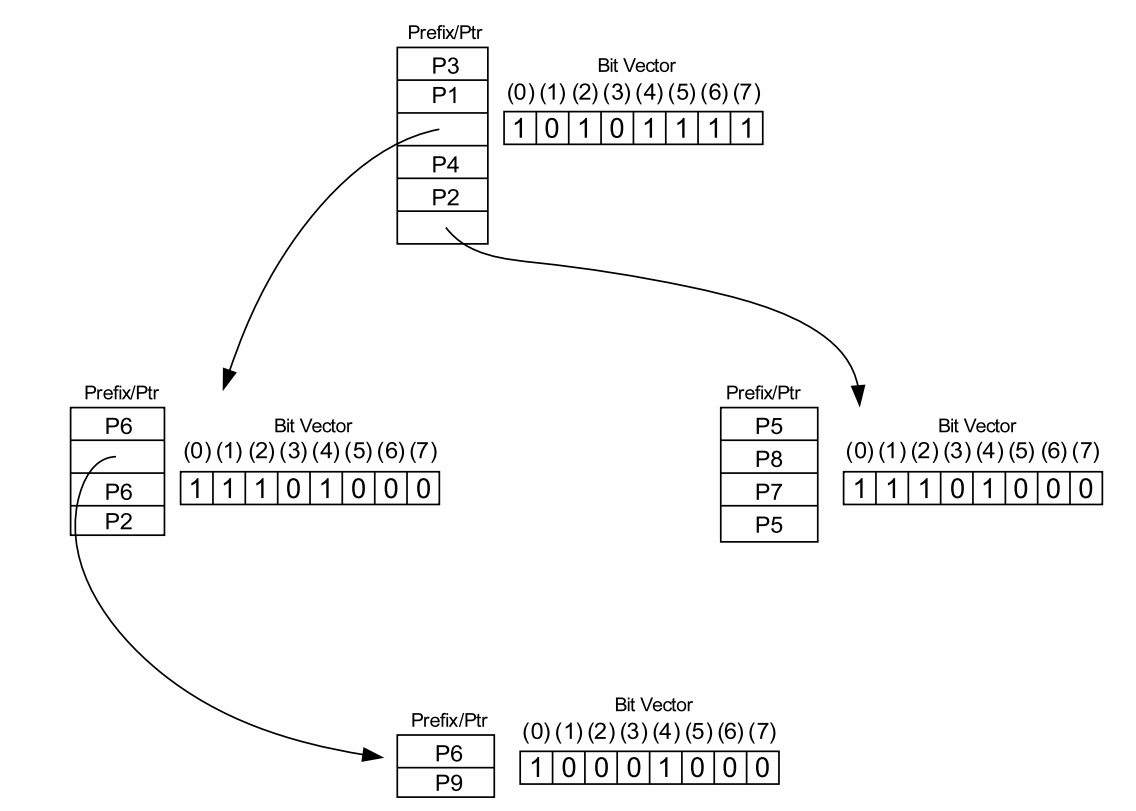

Figure 4.12: *Node compression by bit vector in the Lulea scheme*

and including bit seven, are counted. The result in this case is six, so the contents of location six in the compressed root node are accessed.

Location six in the root node contains a pointer to the right-hand child node. The next three bits of the input, 111, are used as an index into the bit vector of the child node, again bit seven. Counting from the left again, there are four bits set to one, thus location four in the child node is accessed. Location four returns filter P5 - the longest prefix match - and the search terminates. The Lulea scheme requires some 200KBytes to accommodate the Mae-East database, a significant improvement on Controlled Prefix Expansion.

Eatherton et al. [84] propose some further enhancements to the multi-bit schemes just described with their Tree-Bitmap structure. They note that Leaf Pushing makes it virtually impossible to bound system update times (since in the worst case the entire structure may have to be re-written), and eschew it in favour of a novel two-bitmap encoding for each node. With storage requirements of the same order as the Lulea approach, the authors claim support for

25 million lookups per second with on-chip SRAM, and guaranteed update performance - at least 10,000 updates per second. The Tree-Bitmap approach has gained industry popularity, and is implemented in Cisco's 40Gbps CRS-1 Router [92].

4.5 Techniques for Exact Matching

Exact matching poses some interesting challenges. On one hand, the task is simplified by the fact that one need no longer accommodate the complexities of variable length prefixes, and (initially at least) the classification task is restricted to packet headers. On the other, one must now consider application areas where the use of heuristics may be impossible, since one may have no *a priori* knowledge of the traffic profile. Rather than establish efficient classifiers based on a known ruleset in a firewall or router database, one may now be required to classify packets which, in the worst case, may exhibit little or no correlation in the way they appear in the network. Additionally (and particularly in the context of flow monitoring) connections may be set up or taken down much more frequently than say, a firewall filter set would be updated. Thus update complexity becomes increasingly important.

4.5.1 Trees

Tree-based techniques (discussed already in the context of longest prefix match) are also applicable to an exact match search. Bennett [93] has proposed an elegant mapping of one such tree structure into RAM. Consider the simple case of looking up the 32 bit input[4] 0x2E384534 illustrated in Figure 4.13.

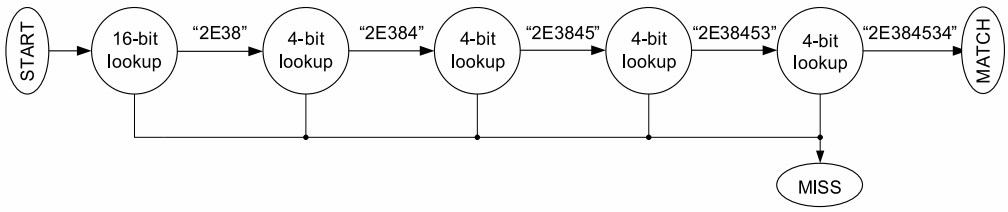

Figure 4.13: *Multi-stage lookup as specified by Bennett*

The basic idea here is not new. The input key is partitioned into sub-blocks of multiple bits, and a partial match is established on one sub-block in each stage of the lookup. A miss is reported

[4] 0x denotes hexadecimal representation

in the event of a mis-match in any of the sub-block lookups. Described by the author as a neural network, the structure is effectively a multi-way trie, with each sub-block lookup a node in that trie. What is more interesting is the way this structure is implemented in hardware, in what the author calls a *scattered memory device* shown in Figure 4.14.

The five RAM blocks have 16 address bits and thus 65,536 addressable locations. Considering again the lookup of 0x2E384534 (and assuming this data is indeed already stored in the system) the first sixteen bits 0x2E38 are used as the address into the Level 1 RAM. The 12 bits stored at this location represent a pre-programmed offset into the Level 2 RAM. This is then combined with the next 4 bits of the input (0x4 in this case) to form the complete pointer into the Level 2 RAM, where another 12 bit offset is found, and the process repeats. Comparators at each level in the search indicate a match or mismatch, the latter terminating the search.

Tal and Itzchak [94] propose another interesting variation on a tree structure for packet classification which they call a *Look-Ahead Tree*. They note that the number of bits of interest in a typical single tree node is significantly less than the number of bits which can typically be transferred in a single memory access. They thus propose an alternative contiguous memory model which allows multiple nodes to be accessed in a single cycle to bridge this gap in efficiency.

4.5.2 Hashing

Processing data through a hash function - *hashing* - is useful in a variety of applications including cryptography - where one seeks to create secure digital signatures for sensitive electronic information; cyclic-redundancy-checking (CRC) - where one seeks to verify the integrity of data which may have been corrupted over time or space; and information retrieval - where one seeks efficient access to stored data via some index into a data structure known as a *hash table*.

4.5.2.1 Simple Hashing for Packet Classification

In general terms, a *hash* function or algorithm is one which transposes or substitutes the bits of some input data to produce a repeatable pseudo-random output. Such functions are *deterministic*; that is if two outputs from such a function differ, one can say with certainty that the inputs which generated these outputs also differ. On the other hand, hash functions are not

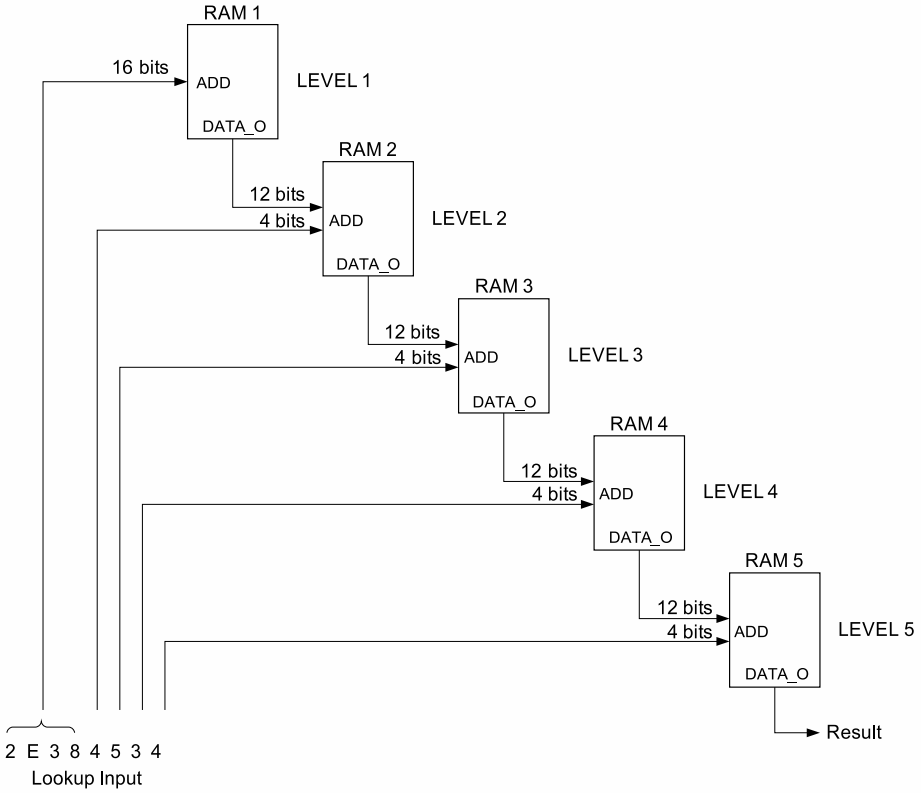

Figure 4.14: *Multi-stage lookup implemented in RAM*

injective; that is if two outputs from such a function are the same, one cannot say with certainty that the inputs which generated the outputs were the same.

Hash functions are typically defined in terms of a *domain* - the possible inputs to the function, and a *range* - the possible outputs. Often hash functions will have an infinite domain, and a range determined by some practical constraint - memory capacity being a typical example. Although it is possible for a hash function to have a domain and range which are the same size (in effect a one-to-one mapping, with a unique output for every possible input), it should become apparent in the discussion which follows that such functions are of limited applicability to packet classification.

Consider a packet classification scheme based on arbitrary 32-bit inputs. There are thus $2^{32} = 4,294,967,296$ possible input permutations, corresponding to over 4Gb of required storage were one to permit a unique memory location for every possible input. Matching the domain and range sizes would require a prohibitive amount of storage. Hashing into a compressed range

offers a more promising way forward. Say, for example, that one uses a function which takes any 32-bit input and generates an 8-bit uniformly random output signature. If one allocates memory storage in a hash table based on this output signature rather than on the input data, one need manage only $2^8 = 256$ memory locations. A perfect hash function will (with perfect uniformity) distribute every possible input in the domain across this 256 location range.

Clearly, there is a catch. One cannot simply take an arbitrarily large domain and compress it for free. In reality collisions occur, as shown in Figure 4.15. Consider a naïve scheme in which the 32-bit input is simply truncated to 8-bits to generate a storage location (ignoring for now the fact that the range distribution of such a scheme would be poor). In the domain, where all 32 bits of the input data are considered, Key 1 (0x1E5822CF) and Key 2 (FDFFDFCF) are clearly different. However, after truncation to 8-bits into the compressed range, Key 1 (0xCF) and Key 2 (0xCF) become indistinguishable, resulting in a memory collision when one tries to use the range index to store the information.

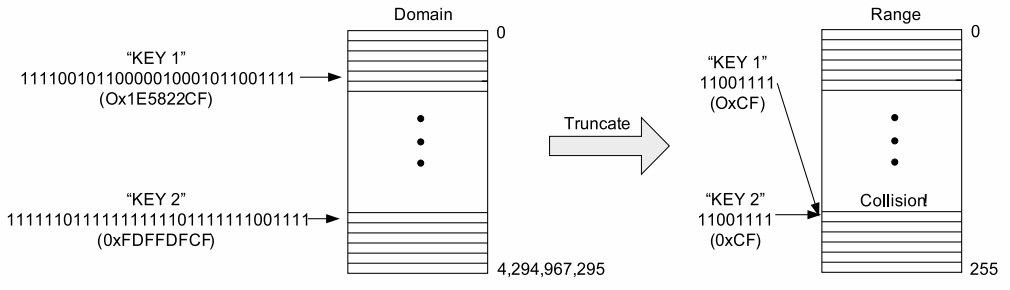

Figure 4.15: *Memory collision in single hashing*

Collision resolution thus becomes an integral part of any hashing based information retrieval. Knuth's ubiquitous techniques of *chaining* and *open addressing* [95] are well documented in the literature. With chained collision resolution, each entry in the hash table is effectively a pointer to a linked-list which contains all of the inputs which collided at that location. The lookup operation thus becomes a traversal of a linked list in some additional memory space reserved for collision resolution, a technique employed in [96] for per-flow bandwidth reservation.

Open addressing schemes include *Linear Probing* and *Double Hashing*. In the former, in the event of a collision in the hash table, one seeks to place the colliding data in the next available location, so if location x is occupied one tries location $x + 1$, then location $x + 2$ etc. This can lead to an undesirable lack of uniformity known as *clustering*. This effect is mitigated

in double hashing by generating a memory offset p using a secondary hash of the input data. In this case, when location x is occupied one tries location $x + p$. Tal and Rachamim [94] propose a similar approach, using two independent hash functions on the input data, one to generate a table address and another to generate a data signature which is stored at that address to accelerate retrieval.

Clearly the performance of any hash-based packet classification scheme is closely coupled with the probability of collision in that scheme, since whatever collision resolution method one implements will require additional memory accesses compared with a collision-free lookup. The design of hashing systems thus poses some classic engineering trade-offs. As the size of the hash function's range approaches the size of its input domain, the chances of collision are reduced (since one is applying minimal compression) - and lookup times are reduced at the cost of memory. Reducing this range saves memory resource, but increases the chances of collision and degrades lookup performance. Similarly, cryptographic strength hash functions will give near perfect random distribution and reduce the likelihood of collision, but will be more difficult and costly to implement than simpler CRC-based equivalents.

4.5.2.2 A Simple Probability Model for Single Hashing

In framing the behaviour of a single hashing system a little more precisely, the frequently used "balls into bins" model is a useful analogy [97]. The basic question is as follows: One starts with m balls and n bins. For each ball, one chooses a bin, uniformly at random, in which to place the ball. When the balls have been exhausted, what can one say about the number of balls in each bin? Or analogously, if one starts with m different packet headers for allocation into a hash table of n entries, what can one say about the number of packet headers associated with each hash table entry when all the headers have been allocated?

Such questions have generated a battery of detailed mathematical analysis, introduced by Gonnet [98] and Larson [99]. From such analyses one gains interesting insight into the loading behaviour of the balls and bins experiment, and by analogy the collision behaviour of a single-hash based classification system. One key result is of particular interest in the discussions which follow and is stated without proof below.

Single hashing produces a binomial distribution in each hash bin, which in the limit is a Poisson distribution. That is, if there are *n* bins and *m* items hashed into them, such that one may define

a *load factor* $a = m/n$, then the probability that a given bin has *k* items in it, $P(k)$ is given by Equation 4.1.

$$P(k) = \frac{a^k e^{-a}}{k!} \tag{4.1}$$

4.5.3 Contemporary Hashing Techniques

Hashing has inspired a significant body of contemporary research in packet classification. Authors have noted that in terms of lookup times, hashing performs very well on average, but poorly in the worst case, when multiple collisions occur and multiple memory accesses are required to resolve the lookup. Whilst the use of strong cryptographic hash functions reduces the likelihood of such collisions, they are difficult to compute within the minimum packet time, typically taking multiple clock cycles to produce a result [100]. An alternative approach proposed by Srinivasan and Varghese [91] is to compute a suitable semi-perfect hash function based on *a priori* knowledge of the entries to be hashed. Such computations can take several minutes, making them impractical for systems with frequent dynamic content updates such as TCP flow monitoring.

More promising are solutions based on multiple hashing, introduced by Broder and Karlin [101], which have been shown to perform better than single hashing. In their multiple table scheme, items which collide in the first table percolate through to a location in a second hash table determined by a second independent hash function; items which collide in the second percolate down to the third, and so on. In the event of an item percolating through the entire available hash space, a partial re-hash of the input items is computed. The authors make the point that such re-hashing is seldom required. Lim et al. extend this approach by using a small CAM, which may be accessed in parallel with the hash tables, to accommodate the items which have overflowed [102].

Also leveraging the advantages of multiple hashing, two contemporary approaches to packet classification appear to perform particularly well. The first approach uses a structure called a *Counting Bloom Filter* and the second, an algorithm for improved load balancing known as *d-left*. These are now considered in more detail.

4.5.3.1 Counting Bloom Filters

A Bloom Filter [103] is a hash-based data structure which indicates whether or not a given item is a member of a set. The basic idea is illustrated in Figure 4.16. The item to be stored is hashed using k independent hash functions. Each hash function generates the address of a bit in a bit vector of length m. Each bit in the vector addressed by a hash function is set, effectively creating a signature for the item and indicating its presence in the set. Item signatures can overlap however, resulting of the possibility of *false positive* results, where one item signature aliases with another.

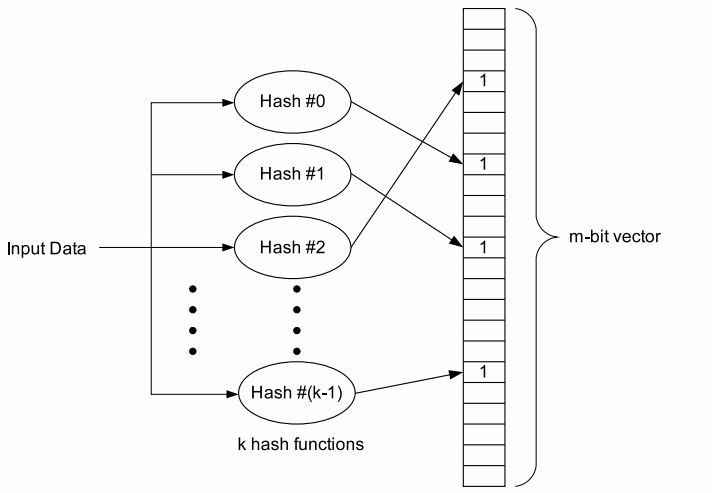

Figure 4.16: *A Bloom Filter*

Bloom Filters have previously been used in both deep packet inspection and longest prefix matching applications [104, 105]. By implementing a Bloom Filter in high bandwidth embedded memory, one can identify very quickly when an item is not present in the set, and thus remove redundant and time consuming searches into off-chip memory. However, as Song et al. [106] point out this does nothing to improve performance in the event that the bloom filter indicates the presence of an item, and an off-chip search is initiated. They thus proposed an improved structure called a *Counting Bloom Filter*.

In this embodiment, which they label a *Basic Fast Hash Table*, each bit in the original vector is replaced by a counter. Upon insertion of a new item, each counter addressed by a hash function produced by the item to be inserted is incremented. To query whether or not an item is already in the set, one computes the requisite hash functions, and tests that all the addressed counters are

non-zero. Consider the simple example of Figure 4.17, with $k = 3$ hash functions addressing an array of $m = 12$ counters.

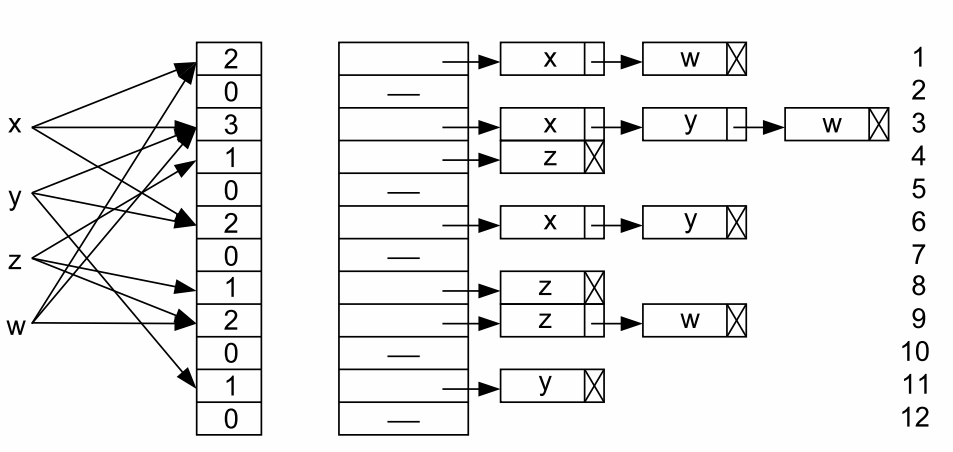

Figure 4.17: *Packet classification with a Counting Bloom Filter*

Items x,y,z and w are inserted sequentially. When a new item arrives, each of the counters addressed by the corresponding 3 hash functions is incremented and the item stored in a linked-list corresponding to that counter. Thus, in this scheme, each of the items is stored in the system k times.

The search procedure for an item is similar. When an item to be queried arrives, one computes the k hash functions and interrogates the counters addressed by each. In the event that all the counters are non-zero one knows (notwithstanding the possibility of a false positive, which can be tuned to be low) that the item is stored in the system. The authors note that by traversing the linked list associated with the counter with the lowest value, one can minimise the number of external memory accesses required to resolve a lookup. For example, again considering Figure 4.17 after all insertions have been completed, if item y is queried, count values of 3, 2 and 1 will be returned from counters 3, 6 and 11 respectively. Choosing counter 11 as the lowest value, y can be retrieved in a single memory access.

The authors go on to propose a number of optimisations, pruning and balancing the structure to reduce storage and offering a probabilistic analysis which indicates that in the worst case, any item may be retrieved in a single memory access with a optimal filter configuration. The hash space is implemented in external DRAM memory, and the counters stored locally in FPGA embedded memory.

4.5.3.2 From d-random to d-deft

Also based on multiple hashing are two related load balancing algorithms known as *d-random* and *d-left*, which combine hashing with an element of choice regarding where an input item is actually allocated. First analysed by Azar et al. [107], d-random splits the n hash table locations into d sections, each addressed by one of d hash functions. The loads in each of the addressed hash table locations are examined, and the current item inserted in the location with the lowest load. Figure 4.18 compares a simple 2-random example with single hashing. In the 2-random case, the current item hashes into location 5 in the left hand table and location 13 in the right hand table. The load (indicated by the circles in the figure) is 2 in both bins. In d-random, such ties are broken arbitrarily.

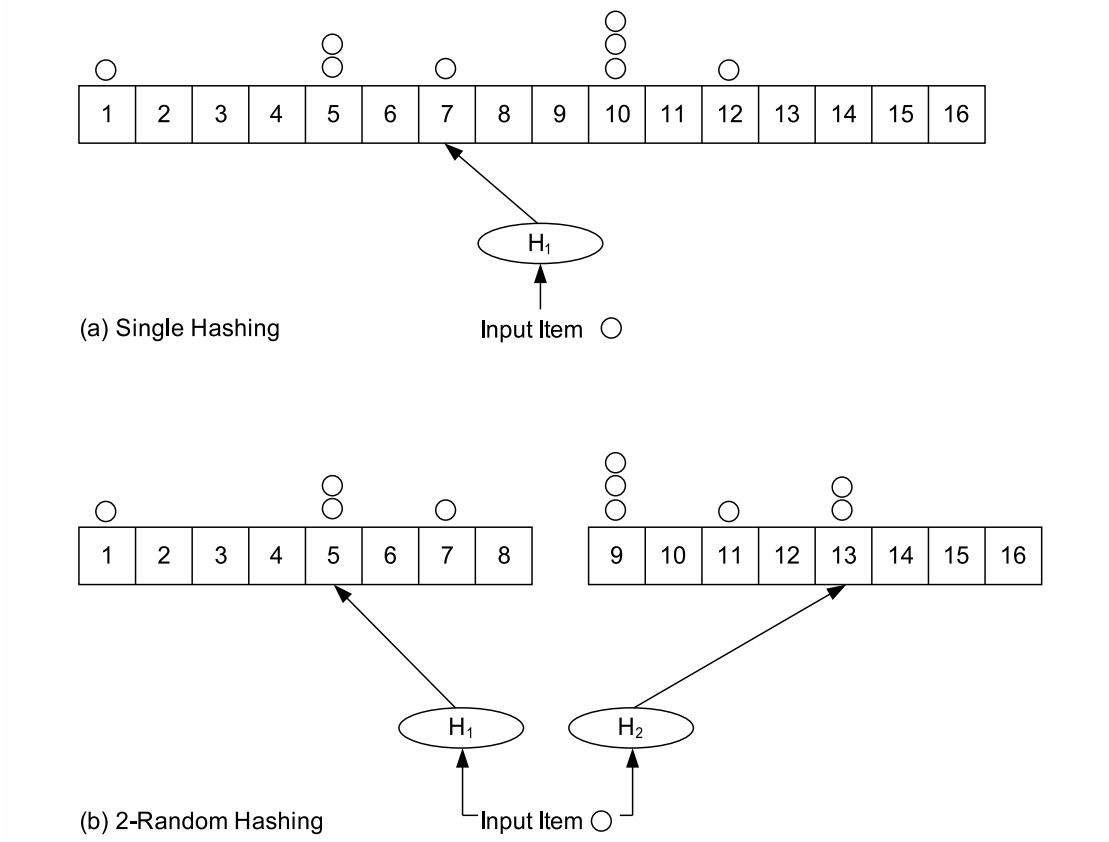

Figure 4.18: *Single hashing (a) and two-random hashing (b)*

The *Always-Go-Left* or *d-left* algorithm first proposed by Vöcking [15] is very similar to *d-random* except that in the case of a tie in the examined load, rather than allocate the item

arbitrarily, it is always placed in the left-most bin. Both *d-random* and *d-left* have been analysed using abstract and complex witness tree arguments [15, 108, 109] which offer bounds on the likely maximum load in the system after a fixed number of items have been allocated, and latterly using differential equations [16, 110–112] to yield numerical loading results for the entire hash table.

Both *d-random* and *d-left* algorithms offer significant performance improvements over single hashing. Even in the simplest embodiments where $d = 2$, *d-random* has been shown to provide an exponential decrease in the maximum observed load [107]. The asymmetry introduced when breaking ties in *d-left* has been shown to improve the load performance still further [112]. This improved load balance has obvious implications for packet classification; fewer collisions mean fewer external memory accesses and faster lookup resolution.

4.6 Towards an FPGA-based Packet Classification Engine

Reiterating the design goals alluded to earlier, the focus of this research is to investigate and design an FPGA-based packet classification engine with primary application focus on exact matching. Ideally one wishes to design such an engine with minimum time, space, power and update complexity - in other words, one wants to be able to classify potentially millions of flows at equipment line rates using reasonable amounts of FPGA resource and external SRAM or DRAM. Additionally, whilst the emphasis remains on exact matching capability for this project, one nonetheless seeks to develop an approach which could migrate to the other lookup paradigms discussed.

Tree structures and hashing schemes appear to offer the most promising ways forward. Trees allow lookup performance to be specified in a completely *deterministic* fashion. For example, if one designs a multi-way tree to lookup keys of 64 bits, with 8 bits compared at each level in the structure, one can guarantee traversal of the entire tree in 8 steps, or external memory accesses, and thus provide a hard upper bound on worst-case lookup times. Therein also lies the weakness in the tree-based approach, since lookup performance is correlated to the length of the input key. Assuming that external memory access widths remain approximately constant, longer input keys mean additional sequential memory accesses per lookup - clearly a problem for system scalability to IPv6, where the additional address space alone means that lookup keys will be almost 300 bits long (compared to 104 for the standard IPv4 5-tuple).

With hashing-based schemes, even when average performance is good, it is virtually impossible to make guarantees about the lookup time in the worst case, since one cannot guarantee collision free operation. Thus hashing is often said to be *non-deterministic*. However, with the advent of advanced hashing techniques as discussed in 4.5.3 one can approach deterministic operation with very high probability. Further, the lookup performance of hashing based schemes is not correlated with input key length; an important benefit in the move to IPv6.

Commercial considerations are also significant in identifying a way forward here, since Aliathon Ltd. must remain free to release products based on whatever classification technology is developed in the course of this research. The patent space is of particular significance. For example, the optimised Tree-Bitmap technique developed by Eatherton et al. has been patented by Washington University [113], and the use of Counting Bloom Filters in classification applications is covered in two patent applications assigned to Global Velocity Incorporated [114, 115].

With all the above points in mind, the *d-left* approach was selected as the best candidate for further investigation. Whilst its applicability to packet classification has been suggested by Broder and Mitzenmacher [16], it has received little further attention in subsequent published work, despite promising initial results. Further, whilst hinting at the suitability of *d-left* in a *dynamic* context (where items are frequently deleted and inserted in the system) the authors leave some interesting questions unanswered.

4.6.1 Understanding d-left: A Numerical Analysis

In the following discussion, the numerical approach introduced by Mitzenmacher [110] and elaborated in conjunction with Broder [16] is followed, since this yields a detailed numerical approximation of the expected loads across the entire hash table, rather than a bound on the expected maximum observed load in the system. As should become apparent, from a hardware design perspective, such numerical results are extremely useful. For convenience, let us initially consider a *d-left* model where $d = 2$, since this is the simplest case to analyse, and the incremental performance gains have been shown to diminish as d is increased further [16].

In this case, n hash bins are divided up into 2 groups of $\frac{n}{2}$ bins each. Let $y_i(t)$ be the fraction of the n hash bins that contain at least i items and are in the left hand group after nt items have been allocated. Similarly, let $z_i(t)$ be the fraction of the n hash bins which contain

at least i items and are in the right hand group when nt items have been allocated. Thus $y_i(t), z_i(t) \leq 1/2$ and $y_0(t) = z_0(t) = 1/2 \ \forall\ t$. If one choses a random hash bin on the left, the probability that it contains at least i items is thus given by $\frac{y_i(t)}{1/2} = 2y_i(t)$. Identically if one choses a random hash bin on the right, the probability that it contains at least i items is $2z_i(t)$. Mitzenmacher incorporates these simple probabilities into two differential equations (for $i \geq 1$) which describe the behaviour of a 2-left system as the number of bins n and the number of allocated items nt approaches infinity.

$$\frac{dy_i(t)}{dt} = 2(y_{i-1}(t) - y_i(t))(2z_{i-1}(t)) \tag{4.2}$$

$$\frac{dz_i(t)}{dt} = 2(z_{i-1}(t) - z_i(t))(2y_i(t)) \tag{4.3}$$

To explain these equations, recall the basic operation of the *2-left* algorithm. The item to be inserted will be hashed using two independent hash functions. The load in each hash bin addressed by the hash functions will be examined and the new item allocated to the hash bin with the lowest load. In the event of a tie, the new item will be allocated left.

Let dt represent the interval of time in which one item is allocated in the hash table. For $y_i(t)$ to increase in this interval, the item to be inserted must choose a bin on the left with *exactly* $i-1$ items (with probability given by $2y_{i-1}(t) - 2y_i(t)$) *and* a bin on the right with *at least* $i-1$ items (with probability given by $2z_{i-1}(t)$). The probability of both these events occurring in the given time interval is simply the product of the individual probabilities, as given in equation 4.2. Similarly, for $z_i(t)$ to increase over the interval dt, the new item must choose a bin on the left with *at least* i items and a bin on the right with *exactly* $i-1$ items, with probability given by the product as given in equation 4.3.

For convenience of notation, Mitzenmacher then proposes combining these two equations in terms of a single sequence $x_i(t)$, where the *even* i terms represent the left-hand table probabilities and the *odd* i terms represent the right-hand table probabilities, such that $y_i(t) = x_{2i}(t)$, and $z_i(t) = x_{2i+1}(t)$. So (for $i \geq 2$) one obtains equation 4.4.

$$\frac{dx_i(t)}{dt} = 4(x_{i-2}(t) - x_i(t))(x_{i-1}(t)) \tag{4.4}$$

So, say one wanted to calculate the fraction of bins in the left hand table with load at least 1, $y_1(t) = x_2(t)$. Substituting into equation 4.4 yields:

$$\frac{dx_2(t)}{dt} = 4(x_0(t) - x_2(t))(x_1(t)) \tag{4.5}$$

Since $x_0 = y_0 = \frac{1}{2} \; \forall \; t$ and $x_1 = z_0 = \frac{1}{2} \; \forall \; t$ then:

$$\frac{dx_2(t)}{dt} = 1 - 2x_2 \tag{4.6}$$

Equation 4.6 is easily solved as an initial value problem since $x_2(0) = 0$, and has a general solution given by:

$$x_2(t) = \frac{1}{2}\left\{1 - e^{-2t}\right\} \tag{4.7}$$

Thus as the number of items allocated tends to infinity, the fraction of the total bins with load at least one which are in the left hand table tends to $\frac{1}{2}$, as one would expect. In similar fashion, one can obtain general solutions for the higher order x_i terms, although as i increases, the nested x_{i-2} and x_{i-1} terms make the arithmetic progressively more difficult to deal with. Behaviour of the terms x_1 through x_7 is shown graphically in Figure 4.19.

Given the initial system scaling, one can then use the x_i terms to analyse the loading at points of interest in any given allocation. For example, the loading profile created by a *2-left* allocation of n items into $2n$ hash bins could be found by evaluating the x_i terms at $t = \frac{1}{2}$, the profile created by an allocation of n items into $4n$ bins by evaluating the x_i terms at $t = \frac{1}{4}$ and so on.

Clearly it is impossible to derive a complete analytical solution here, since the x_i vector is infinitely long. However, the analysis presented in [16] shows that for practical 2-left systems, the fraction of bins with load of least i falls extremely quickly with increasing i (in fact approximately as $2^{-2.6^i}$) such that high loads are unlikely. Thus, one may truncate the x_i vector and still obtain very accurate loading predictions for systems of practical interest. Furthermore, one can avoid the complexities of the higher order x_i terms by generating numerical, rather than analytical, solutions to the differential equations.

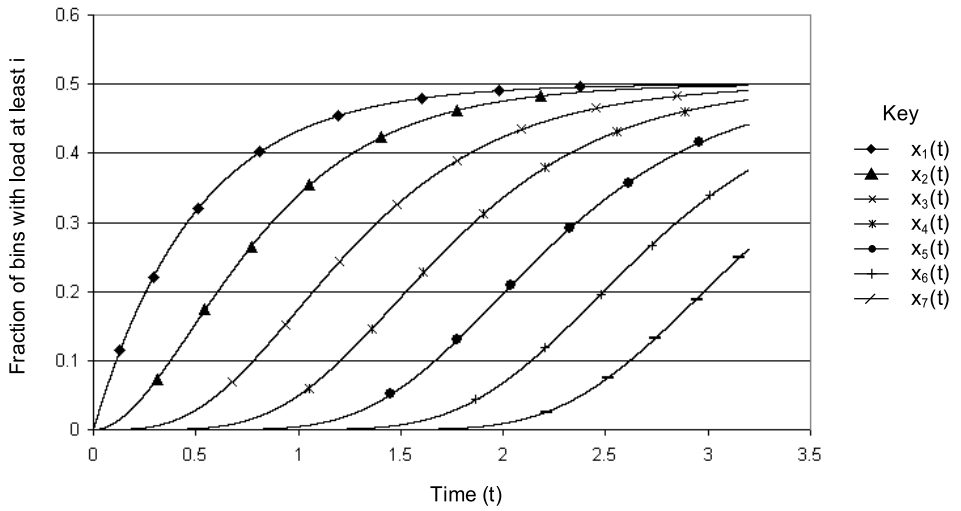

Figure 4.19: *Expected behaviour of the $x_i(t)$ terms from the 2-left differential equations*

The Euler and Runge-Kutte approximations [116] are classical methods for obtaining numerical solutions to differential equations where general solutions are impossible or difficult to find. The former is used to generate the numerical results in this research.

4.6.2 Existing Numerical Results

In [16] the authors present some promising comparative results, illustrating the performance gains of the *d-left* approach over single hashing. Table 4.3 shows these results for the *2-left* case, where entries represent the fraction of the hash bins with load *exactly* i. For the *2-left* case, these results have been generated numerically, using Equation 4.4. For the single hashing case, one can establish the expected loading from Equation 4.1. Note that in the *2-left* equations, although the x_i terms reflect the fraction of bins with load *at least* i, by simple subtraction one can calculate the fraction of bins with load *exactly* i. For example, if the fraction of bins with load at least 3, $x_3(t) = 0.45$ and the fraction of bins with load at least 4, $x_4(t) = 0.27$ then the fraction of bins with load *exactly* 3 is given by $x_3(t) - x_4(t) = 0.45 - 0.27 = 0.18$.

Recalling that the ratio of the number of items to be hashed m to the number of available hash bins n is the *load factor* $a = \frac{m}{n}$, one can see from Table 4.3 that in both cases, as the load factor decreases, the loading performance improves. Intuitively one might expect this since a more sparsely populated hash table will obviously produce fewer collisions (or bins

Number of Items

Load i	n/2	n	2n	3n	4n
1	3.0e-01	3.7e-01	2.7e-01	1.5e-01	7.3e-02
2	7.6e-02	1.8e-01	2.7e-01	2.2e-01	1.5e-01
3	1.3e-02	6.1e-02	1.8e-01	2.2e-01	2.0e-01
4	1.6e-03	1.5e-02	9.0e-02	1.7e-01	2.0e-01
5	1.6e-04	3.1e-03	3.6e-02	1.0e-01	1.6e-01

(a)

Number of Items

Load i	n/2	n	2n	3n	4n
1	4.4e-01	5.5e-01	2.1e-01	4.0e-02	6.9e-03
2	3.0e-02	2.2e-01	5.0e-01	2.0e-01	4.3e-02
3	8.6e-06	4.4e-03	2.6e-01	4.8e-01	1.9e-01
4	9.2e-16	5.2e-08	9.1e-03	2.7e-01	4.7e-01
5	1.4e-42	1.2e-21	5.0e-07	1.2e-02	2.8e-01

(b)

Table 4.3: *Expected fraction of bins with load exactly i with variable number of items allocated into n bins by single hashing (a) and 2-left hashing (b)*

with multiple loads). However, relative to single hashing, and particularly apparent as the load factor becomes more favourable, it may also be seen from Table 4.3 that the loading falls off much more quickly when using *d-left*. For example, with a load factor of $\frac{1}{2}$ (that is, allocating $\frac{n}{2}$ items into n bins) the fraction of the bins with load exactly 5 is 1.6×10^{-4} when using single hashing, but only 1.4×10^{-42} when using *2-left*.

The authors of [16] illustrate how these numerical results correlate with real systems by simulating the underlying allocation process in software. As an example, they consider the case of allocating 32,000 items into 32,000 bins. The numerical results predict that 5.2×10^{-8} of the bins will have load 4 or greater. Thus over say, 10,000 runs (where in each run, 32,000 items are allocated) one would expect to see around 16 or 17 bins with load 4. The authors in fact observe a maximum load of 4 in 14 of the 10,000 simulated allocations.

4.6.3 Numerical Results at Improved Load Factors

In [16] the authors consider results for a limited number load factors down to only $\frac{1}{2}$. A natural question to ask is how a *d-left* allocation performs at more favourable load factors. This was chosen as the starting point for further investigation. Using an Improved Euler numerical method with an approximation interval of 0.0005, the numerical results published in [16] were successfully replicated. By the same method, an additional set of results for load factors of $\frac{1}{3}, \frac{1}{4}, \frac{1}{8}, \frac{1}{32}$ and $\frac{1}{64}$ were computed. A comparison with single hashing is again obtained using Equation 4.1 and the results are shown in Table 4.4.

Number of Items

Load \ i	n/64	n/32	n/8	n/4	n/3
1	0.0154	0.0303	0.1103	0.1947	0.2388
2	1.2e-04	4.7e-04	0.0069	0.0243	0.0398
3	6.3e-07	4.9e-06	2.9e-04	0.0020	0.0044
4	2.4e-09	3.9e-08	9.0e-06	1.3e-04	3.7e-04

(a)

Number of Items

Load \ i	n/64	n/32	n/8	n/4	n/3
1	0.0156	0.0312	0.1245	0.2445	0.3181
2	5.8e-08	9.1e-07	0.0002	0.0027	0.0076
3	3.8e-23	1.5e-19	1.9e-12	5.2e-09	1.2e-07
4	1.0e-63	8.1e-54	3.1e-34	9.8e-25	6.4e-21

(b)

Table 4.4: *Expected fraction of bins with load exactly i with variable number of items allocated into n bins by single hashing (a) and 2-left hashing (b) at improved load factors*

These results show that at more favourable (lower) load factors, the fraction of bins with higher loads in a *d-left* allocation is dramatically lower than when using single hashing. For example, with a load factor of $\frac{1}{64}$ (that is, allocating $\frac{n}{64}$ items into n bins) the fraction of the bins with load exactly 4 is 2.4×10^{-9} when using single hashing, but only 1.0×10^{-63} when using *2-left*.

The improved load factors discussed here are of direct practical significance, since it appears that hash table capacities of practical interest are readily supported by current generation memory components. For example, a system capable of classification on $64,000$ unique IPv4 5-tuples of 104 bits each, allowing a load factor of $\frac{1}{8}$ would require $64,000 \times 104 \times 8 = 53.25$Mbits of RAM. Such a hash table could be implemented using 2 32Mbit QDR-II SRAM devices. Similarly, a system with identical 5-tuple capacity, but a load factor of $\frac{1}{64}$ to further reduce the expected number of collisions could be implemented using 2 256Mbit DDR-II DRAM devices.

So how does one resolve the collisions that remain? The numerical results suggest that with an appropriately chosen load factor, the number of colliding entries in a *d-left* hash table will be small. This raises some interesting possibilities for an FPGA-based *d-left* classifier. More specifically, a natural question is - given advances in high-bandwidth FPGA embedded memory technology (up to 12Mbits of embedded block RAM is available in emerging devices [117]) and an appropriately chosen *d-left* topology to reduce the number of collisions, might it be possible to resolve all these collisions on-chip, and thus create a lookup mechanism offering the advantages of both determinacy and storage efficiency? The discussion which follows is an attempt to answer this question.

4.6.4 From Static to Dynamic Systems

All the previously published numerical results based on analysis using differential equations have been for what might be termed *static allocations*. That is, one starts with a finite number of items to be hashed m, and allocates these items into a finite number of hash bins n, after which the process terminates. One then examines the load distribution in the system. This model is useful in illustrating the performance improvements over single hashing, and is a reasonable analogy for router or firewall applications where system updates are infrequent and can be handled by re-hashing. In applications such as TCP flow monitoring however, flows may be set up and destroyed frequently, such that the dynamic performance of the system is critical. One must therefore consider how to better define and model such a *dynamic* system.

Existing analyses do not provide the numerical results required to determine the feasibility of a dynamic *d-left* system in FPGA hardware. In [16] the authors propose the following dynamic system. Up to some arbitrary point in time t_s, only the insertion of items occurs. For $t < t_s$ the system is thus described (as before) by Equation 4.4. After t_s, insertions and deletions vary. The probability that an event is an insertion is p, and the probability that an event is a deletion is $1 - p$. Items to be deleted are chosen uniformly at random from all items. The authors thus modify Equation 4.4 to account for deletions by noting that the total number of items in the system is given by $\sum_{j\geq 0} j(x_{2j} + x_{2j+1})$, and that the number of balls that can be deleted that cause a reduction in x_i is $\left\{Floor\left\{\frac{i}{2}\right\}\right\}(x_i - x_{i+2})$, where *Floor* indicates "the largest integer less than". Hence the equation describing the behaviour of the x_i terms comprises an insertion and deletion term as follows:

$$\frac{dx_i}{dt} = 4p(x_{i-2} - x_i)x_{i-1} - \frac{(1-p)\left\{Floor\left\{\frac{i}{2}\right\}\right\}(x_i - x_{i+2})}{\sum_{j\geq 0} j(x_{2j} + x_{2j+1})} \tag{4.8}$$

In [16] the authors consider lookup systems where all the items (including those which collide) are nominally stored in *external* hash memory. The contents of external memory are read into a line in *internal* local cache, where comparison with the queried data occurs. Failure in such a system occurs when collisions cause the data fetched from external memory to exceed the capacity of the local cache line. The *maximum* load in the system is thus the critical metric.

In an illustrative example, the authors consider initially allocating 32,000 items into 16,000 bins using 2-left, and thereafter either deleting or inserting an item, each with equal probability, until

either a load of 6 is observed or 10,000,000 steps (where a step is an insertion or a deletion) are completed. They observe that the (numerically predicted) fraction of bins with load at least six after the *static allocation* of items is only 7.2×10^{-19}, so that the system should absorb a large number of subsequent insertions and deletions before a load of 6 is actually observed. In fact, over 100 trials, 10,000,000 steps were completed 75 times, although the smallest number of steps before a load of 6 was observed in a trial was 121,805.

The authors go on to note that the maximum number of items one expects to be in the system appears to be a significant factor in dynamic operation. Similar analyses of dynamic systems have been presented in [15, 109], suggesting that if the number of items in a dynamic system is bounded, then the maximum load in that system is also bounded, with high probability. Specifically, from [15] the following is stated for an infinite sequence of insertions and deletions:

> Suppose that at most $h \cdot n$ items exist at any point in time. Then the always-go-left (d-left) algorithm yields maximum load $\frac{ln(ln\ n)}{d \cdot ln\phi_d} + O(h)$, with high probability at any fixed time step t

Analogous arguments are presented in [108] in the context of minimising congestion in circuit-switched networks - the maximum observed load in the system is again of critical concern here. However, in establishing the feasibility of an FPGA-based *d-left* system where collisions are resolved internally, the maximum observed load is only part of the picture.

4.6.5 Overflow Sufficiency

The system proposed here will enforce physical separation between external memory for the primary hash space, and embedded internal FPGA memory which will be used to resolve collisions. The basic idea is as follows: Any items which collide with an existing entry in the external hash space will be stored in the embedded memory. On lookup, the external memory and embedded FPGA memory will be searched in parallel. Since one only ever stores one item per hash location in external memory, the lookup operation will require only one external memory access time[5]. Although there may be multiple items to resolve in embedded memory, this will be offset by much faster internal memory access times. The critical metric in this

[5]For 2-left, there will of course be two memory accesses, one left and one right, but these can be done in parallel.

system is thus not the *maximum* load in any hash bin, but the *total* number of items which end up in FPGA memory.

Bins with load 0 or load 1 are of no significance here - since these bins are either empty or have an entry which is physically stored in external memory, and requires no internal resource. However, bins with load 2 require 1 entry to be stored in external memory, and a colliding entry to be stored in embedded memory, bins with load 3 require 1 entry to be stored in external memory and 2 colliding entries to be stored in embedded memory, and so on. The cost, in terms of embedded memory resource, of all these bins of higher load becomes the critical design criterion. One must ensure that the available embedded memory capacity is never exceeded or information will be lost and the system will fail. In the context of single hashing Norton and Yeager [118] describe this criterion as *overflow sufficiency*.

4.6.6 A New Study of Dynamic Systems

The *static* behaviour of *d-left* and the apparent boundedness of the *maximum* load when the total number of items is bounded in the dynamic case, suggest the intriguing possibility that overflow sufficiency may be achievable in FPGA memory. However none of the analyses previously discussed offer any insight into the fraction of bins in the system which require the support of embedded memory. An alternative numerical model of a dynamic *d-left* system is proposed here. Without significant loss of generality, one can restrict the dynamic behaviour of the system to produce numerical results which indicate the required embedded memory capacity for a given external load factor.

Once again, a *2-left* system is considered first, since this is the simplest to analyse. The key simplification is to place a hard upper bound M on the maximum number of items ever present in the system. Initially, M items are allocated using *2-left* such that the system is full. Then, rather than considering an arbitrary sequence of insertions and deletions and their respective probabilities, since there is a hard upper bound on system capacity, new items may only be allocated when existing items are deleted. A simple way to model this is to assume that at the upper bound M, existing items are removed (uniformly at random) and replaced by new *d-left* insertions, one at a time, such that the total number of items in the system remains constant. This would seem a reasonable approximation of worst-case behaviour. The equation modelling the dynamic behavior of the x_i terms now simplifies to:

$$\frac{dx_i}{dt} = 4(x_{i-2} - x_i)x_{i-1} - \frac{\left\{Floor\left\{\frac{i}{2}\right\}\right\}(x_i - x_{i+2})}{M} \tag{4.9}$$

4.6.7 Numerical Results for Dynamic Systems

Keeping the physical implementation in mind, from Table 4.4 it may be noted that after the static allocation of items the majority utilisation of embedded memory locations will be due to hash bins containing two items. For example, consider the static allocation of 1000 items into 4000 bins using *2-left*. Since the load factor in this case is $\frac{1}{4}$, one would expect on average $0.0027 \times 4000 = 10.8$ of those bins to contain 2 items after a static allocation. Thus for a typical allocation, 10 or 11 embedded locations are required to support the bins of load 2. Since the x_i terms fall off so quickly in *d-left* the equivalent utilisation for bins of load 3 is only $5.2 \times 10^{-9} \times 4000 = 0.00002$ such that on average, there are no bins of load three and thus no cost in terms of embedded memory. Given that the majority utilisation of embedded memory in the static case is attributable to bins of load 2, let us assume that this remains the case in dynamic systems and neglect any contribution from bins of higher load - seeking to validate this assumption later using simulations of the underlying random processes; for now, this keeps the volume of data generated by numerical simulation manageable.

The system behaviour as $t \to \infty$ is approximated as follows. For load factors of $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}$ and $\frac{1}{64}$ the loading after static allocation is obtained by running Equation 4.4 for $t \leq t_s$ where t_s represents the time at which system capacity is reached. The x_i values at $t = t_s$ are then used as the initial conditions for the dynamic deletion and insertion phase, governed by Equation 4.9 for $t_s < t \leq t_t$ where t_t is some arbitrary time at which the numerical approximation terminates. A new set of numerical results were generated in exactly this fashion. The x_i values were post-processed to yield the fraction of bins with load exactly 2 as the simulation progressed, revealing an interesting interaction between the *d-left* insertion of new items, and the uniform random deletion of items at the capacity bound M - the system appears to reach a steady-state.

Figure 4.20 shows the behaviour for a system with load factor $\frac{1}{4}$. The loading after the static allocation is obtained by approximating Equation 4.4 up to $t = 0.25$ indicating that 0.0027 of the bins will have load 2 at this point. After continuous insertion and deletion at the system limit, the fraction of bins with load 2 initially increases (creating an s-shaped profile), but then reaches an apparently steady-state value - approximately 0.0062 in this case. The measurements were repeated for systems of differing load factors, shown in Figure 4.21. Note that since each

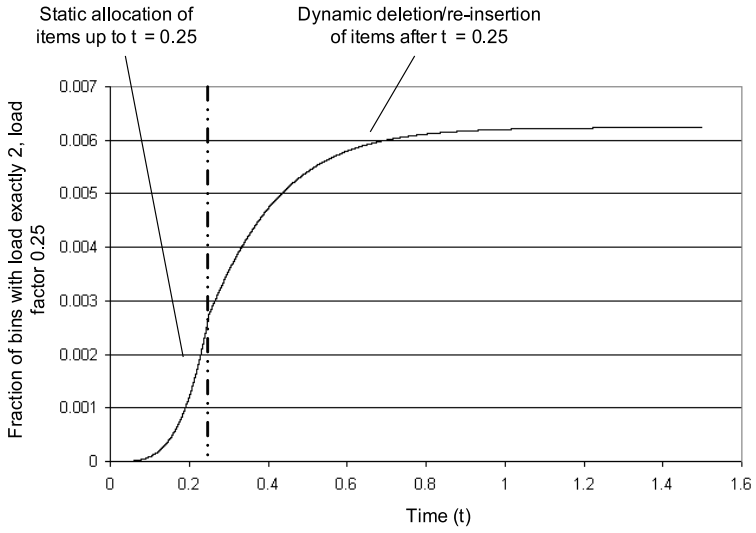

Figure 4.20: *Fraction of hash bins with load exactly 2 for a load factor of* $\frac{1}{4}$*, with deletions/reinsertions at the system capacity* M

halving of load factor yields approximately a ten fold improvement in the steady-state loading, a logarithmic scale is used.

The new numerical results are promising, suggesting that the proposed FPGA-based classifier may be viable even for very large systems. For example, say one is required to accommodate classification of 1,000,000 flows in a dynamic system. With a load factor of $\frac{1}{64}$ one needs 64,000,000 hash bin locations - implementable in DDR-DRAM. From Figure 4.21 the maximum number of bins one would expect to have load 2 is given by $2.23982 \times 10^{-7} \times 64,000,000 = 14.3$, such that one nominally requires only 14 or 15 embedded memory locations to support the system.

4.6.8 Software Simulation of Dynamic 2-Left Systems

The authors of [16] note that (based on number theory dating back to Kurtz [119]) the probability of deviating significantly from the loads given by the differential equations falls exponentially in the size of the system in terms of the number of hash bins, n. By inference then, for practical systems of finite size, one would expect some degree of variance in the predicted load. To investigate this variance, and test the validity of the starting assumption (namely that for systems of practical interest, bins with load 3 or more have negligible impact on the required embedded memory resource) a system simulation was written in C++.

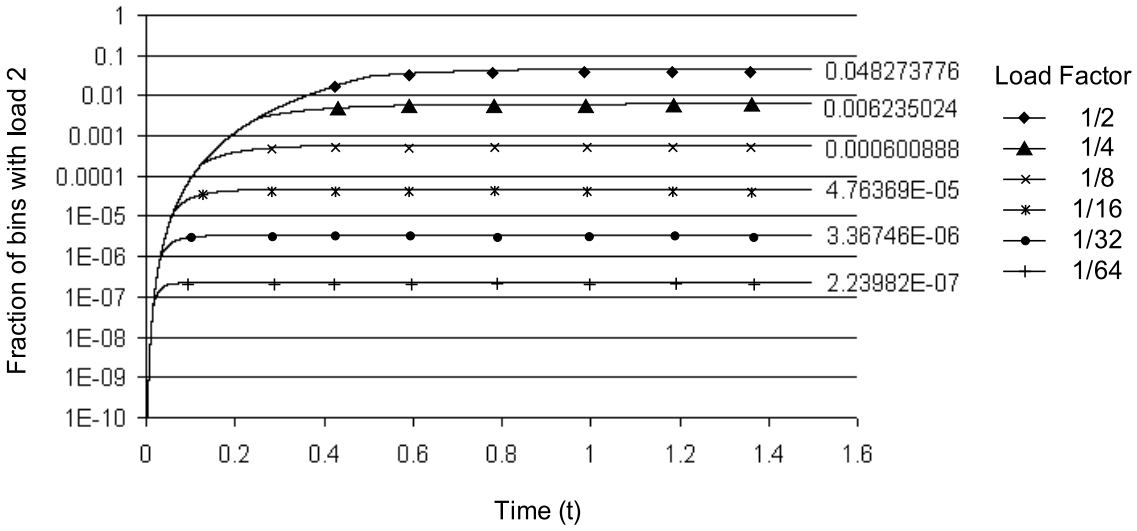

Figure 4.21: *Fraction of bins with load 2 for varying load factor, with continuous deletion and reinsertion at the system capacity M*

The simulation generates 32-bit keys, representative of packet headers, which may be randomised or sent sequentially to the *d-left* classifier. Note that although IPv4 keys are typically 104 bits long, key size is not critical here, since one seeks merely to understand the distribution of the loads. The simulation uses 2 table-based Cyclic Redundancy Check (CRC) functions [120] to generate hash locations for item allocation, and an open-source random number generator for uniform random deletion of items. The simulation monitors the loads in every hash bin, and in real time computes the embedded memory cost to support the system. Note that unlike the numerical approximations previously discussed, the system simulation accounts for bins with load higher than 2. The embedded memory requirements are periodically logged in an output file. To keep data volumes and simulation execution times reasonable, only load factors of $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$ and $\frac{1}{16}$ were modelled. The first set of results, shown in Figure 4.22 reflect a system containing 16,384 items, with a variable number of hash bins. The second set of results shown in Figure 4.24 reflect a system of 262,144 hash bins, with a variable number of items. The predicted embedded memory requirement from the corresponding numerical simulation is shown as a dashed line on each subfigure.

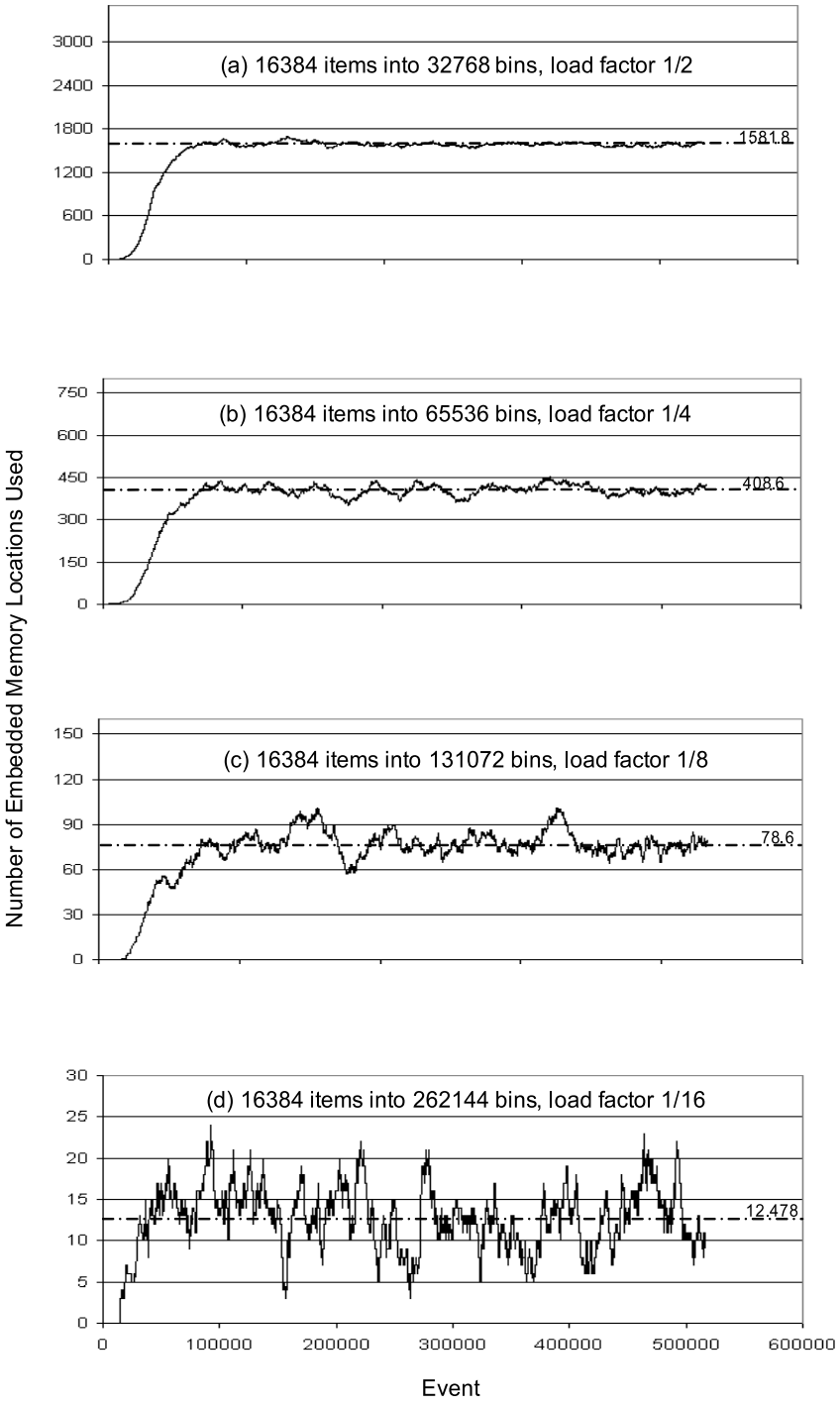

Figure 4.22: *Embedded memory utilisation for a 2-left classifier with 16384 items and variable bins, under dynamic deletion and insertion of items*

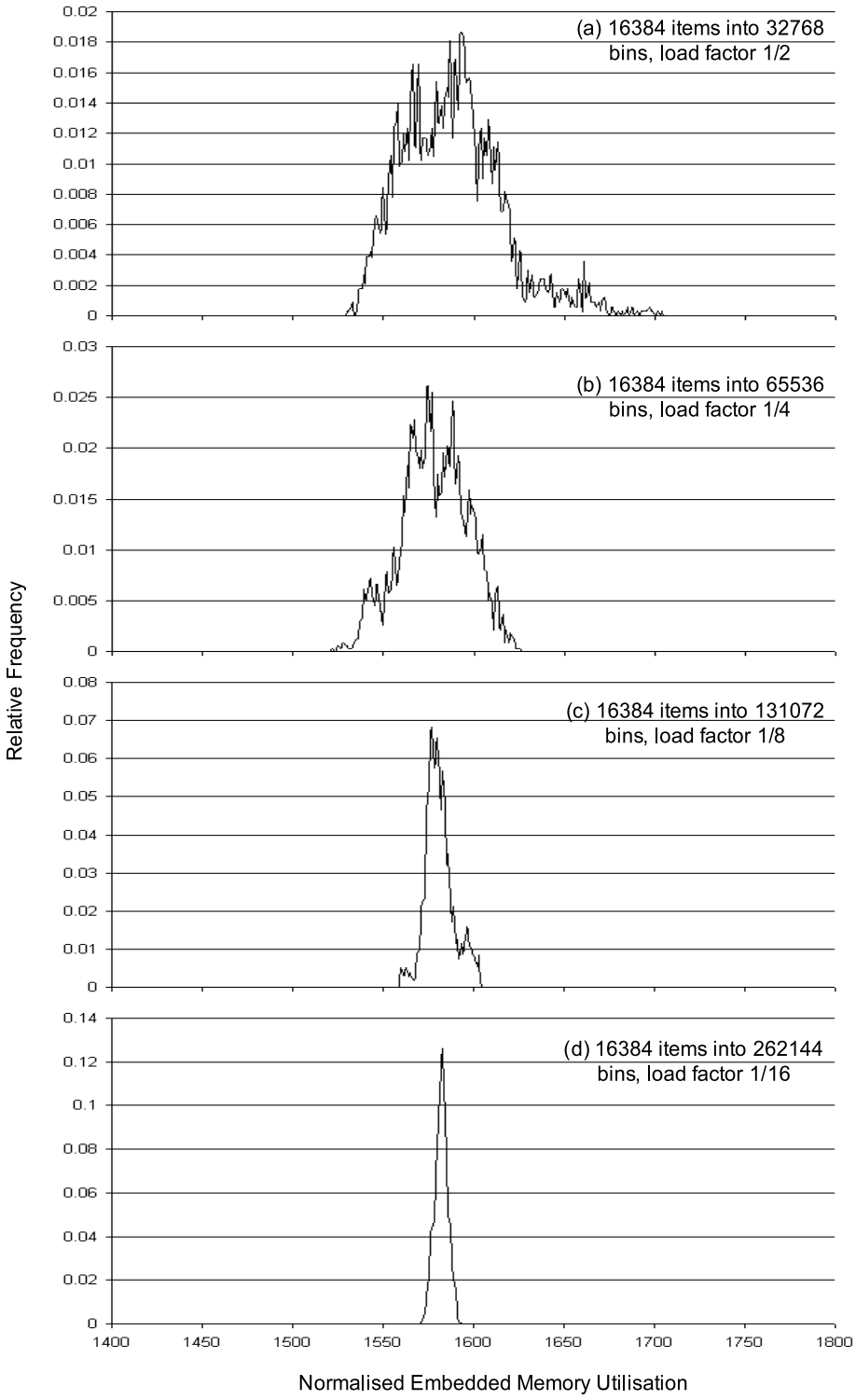

Figure 4.23: *Normalised variance from numerically predicted load for the systems shown in Figure 4.22*

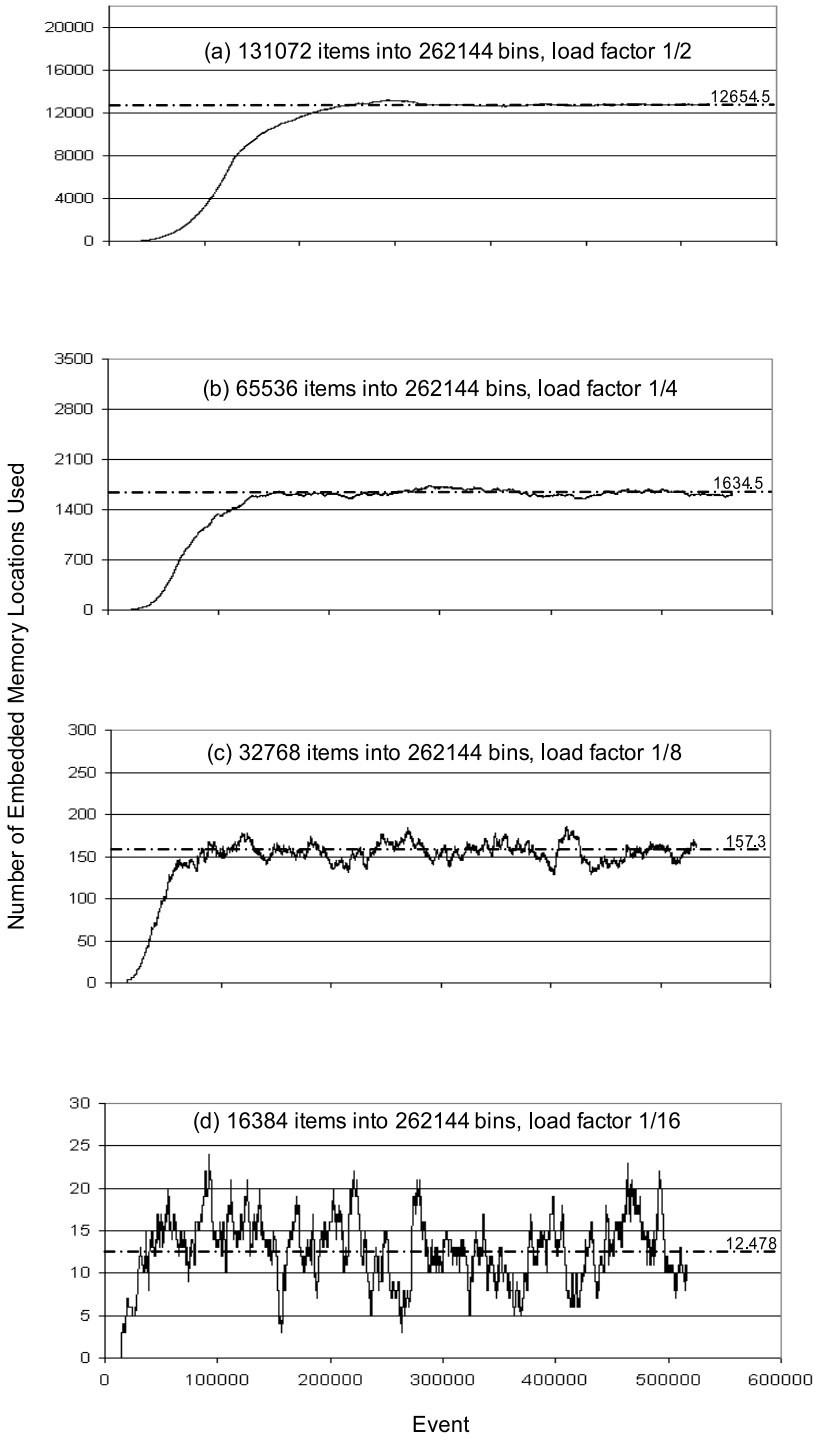

Figure 4.24: *Embedded memory utilisation for a 2-left classifier with 262144 bins and variable items, under dynamic deletion and insertion of items*

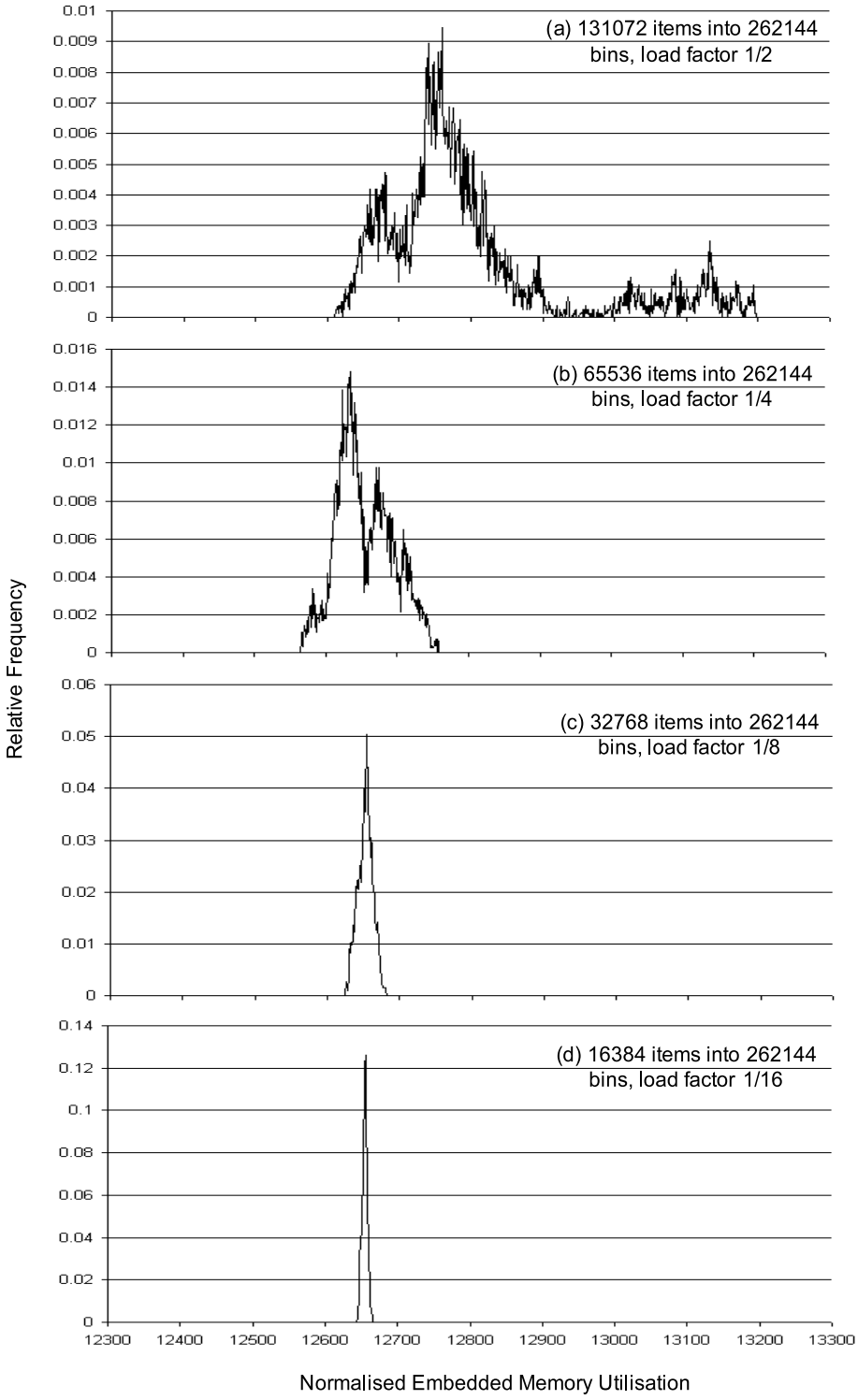

Figure 4.25: *Normalised variance from numerically predicted load for the systems shown in Figure 4.24*

4.6.9 Interpreting the Software Simulation Results

Some care is required in interpreting Figures 4.22 and 4.24. They serve to illustrate only the relationship between the dynamic behaviour and steady state value reached in a real simulation of the underlying random processes, and their equivalents predicted by the numerical model. Since the embedded memory cost falls off so rapidly with improving load factor, the *y-axis* in each of the graphs must be appropriately scaled such that the behaviour in transition from static to dynamic operation may be observed. Thus, in Figures 4.22 and 4.24, although the embedded memory cost for load factor $\frac{1}{16}$ (subfigure(d)) appears to exhibit more variance than that for load factor $\frac{1}{2}$ (subfigure(a)), this is merely an artefact of scaling - in fact the opposite is true; the variance in embedded memory cost improves as the load factor reduces, as illustrated in the corresponding relative frequency plots, Figures 4.23 and 4.25.

4.6.9.1 Characterising Variance

Figures 4.23 and 4.25 comprise a set of normalised relative frequency curves. These are readily constructed by postprocessing the data used in Figures 4.22 and 4.24 respectively. When steady state behaviour is observed, the embedded memory cost output is periodically sampled to construct the corresponding relative frequency distribution. A loading offset is added such that these curves are normalised around the expected mean value for a system of load factor $\frac{1}{2}$ so the distribution shapes may be more easily compared.

As a specific example, consider how Figures 4.22 and 4.23 relate for systems of load factor $\frac{1}{2}$ and $\frac{1}{16}$ respectively. With 16384 items and 32768 hash bins (load factor $\frac{1}{2}$) the steady state embedded memory cost predicted by Equation 4.9 is approximately 1581, shown by the dashed line in Figure 4.22(a). With 16384 items and 262144 hash bins (load factor $\frac{1}{16}$) the equivalent cost is approximately 12, shown by the dashed line in Figure 4.22(d). To make like-for-like comparison of the variance around these predicted means easier, an offset of $1581 - 12 = 1569$ is added to each point in the load factor $\frac{1}{16}$ relative frequency curve, Figure 4.23(d). In other words, although the embedded memory cost for a load factor of $\frac{1}{16}$ varies around a mean of 12 for this system, the distribution is shifted up the *x*-axis for comparative purposes. Equivalent offsets are added for the other load factors modelled, so that they may be compared on the same scale.

From these plots it may readily be seen that choice of load factor is important in establishing

well bounded embedded memory costs. The variance around the numerically predicted mean in the observed embedded memory cost is notably more spread out at higher load factors. In particular, an undesirable high-side roll-off is observed at load factor $\frac{1}{2}$. At lower load factors the situation improves significantly. For example, as shown in Figure 4.25, with 262144 hash bins and 131072 items (load factor $\frac{1}{2}$) the difference between the highest and lowest observed embedded memory cost at steady-state is approximately 600. With 262144 hash bins and 16384 items, the difference between the highest and lowest observed embedded memory cost at steady-state is approximately 30. This is consistent with the number theory alluded to in 4.6.8, and suggests that for large systems, where load factors down to $\frac{1}{64}$ are realisable in DRAM, variance will not significantly impact the efficiency of the proposed implementation.

4.6.9.2 Comparison with Single Hashing

A comparison with dynamic systems based on single hashing yields very favourable results. For each of the load factors previously discussed, an equivalent C++ simulation based on single hashing was constructed. Interestingly, with single hashing, during the dynamic phase of the simulation steady state behaviour is reached immediately. This is what one might expect, since the single hash function represents (approximately) uniform random allocation of items, and the deletion mechanism in the simulation is a uniform random number generator. Hence in the dynamic phase each "event" is a uniformly random insertion followed immediately by a uniformly random deletion - with no net effect on the embedded memory requirements. The numerically predicted steady-state embedded memory cost for dynamic operation of single hashing systems is thus easily obtained from the basic allocation predictions - given in Equation 4.1.

Note that since the number of bins of load higher than two does not fall away so quickly with single hashing, these bins of higher load must be considered when calculating the embedded memory costs. Figure 4.26 shows the steady state behaviour just described for a single hashing system with 32768 items and 262144 hash locations. In this case, From Equation 4.1 it may readily be shown that approximately 1808 hash bins will contain two items, and that 76 hash bins will contain three items. Since for each of these bins of load three, two items must be stored on-chip in the proposed implementation, the embedded memory cost is 76 x 2 = 152. Contribution from bins of load four and higher is negligible. The total numerically predicted embedded memory cost for the system (shown by the dashed line in Figure 4.26) is thus

approximately $1808 + 152 = 1960$, which again correlates well with the steady-state value reached in the system simulation.

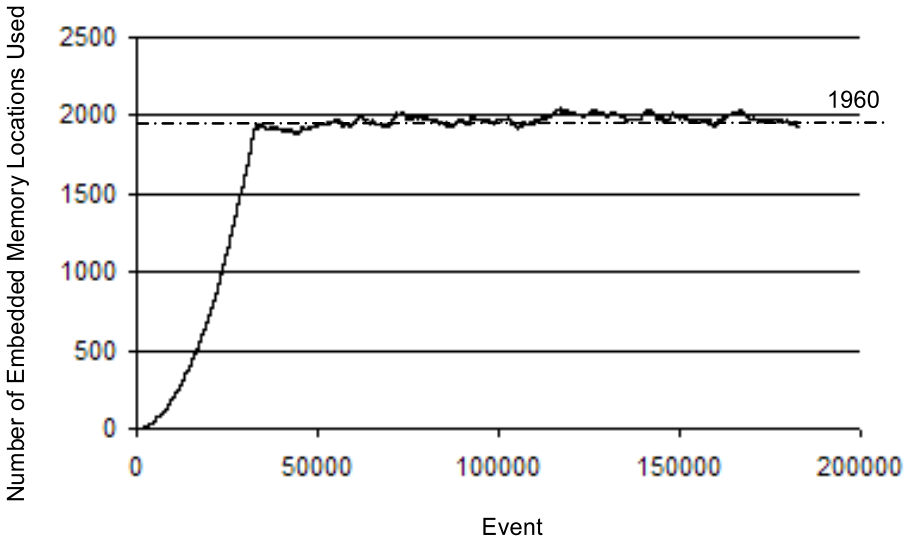

Figure 4.26: *Embedded memory utilisation for a single hashing classifier with 32768 items and 262144 bins, under dynamic deletion and insertion*

Correlating the results for the set of systems considered, one may see that since bins of higher load fall away much more quickly when using *2-left*, it outperforms single hashing - dramatically so at better load factors. Measured in terms of the embedded memory cost required to support the system during dynamic operation *2-left* is approximately 2.2 times better at load factor $\frac{1}{2}$, 4.6 times better at load factor $\frac{1}{4}$, 12.5 times better at load factor $\frac{1}{8}$ and 41.8 times better at load factor $\frac{1}{16}$, as summarised in Table 4.5.

4.6.9.3 Interim Conclusions

In interim summary, the correlation between the embedded memory requirements predicted by numerical approximation and those observed in the simulation of the underlying random processes appears to be excellent. The combination of the loading plot during the static and dynamic phases of operation, and the relative frequency distribution of the embedded memory cost during the dynamic phase provide a reasonably complete picture of system behaviour. From these one may observe that a *2-left* system initially exhibits the predicted "s-shaped" transition during dynamic operation and eventually reaches a steady-state value consistent

Simulation	Required Embedded Memory Locations (2-Left)	Required Embedded Memory Locations (Single Hash)
16384 items into 32768 bins (Load Factor 1/2)	1582	3483
16384 items into 65536 bins (Load Factor 1/4)	409	1883
16384 items into 131072 bins (Load Factor 1/8)	79	985
16384 items into 262144 bins (Load Factor 1/16)	12	499
131072 items into 262144 bins (Load Factor 1/2)	12655	28162
65536 items into 262144 bins (Load Factor 1/4)	1634	7520
32768 items into 262144 bins (Load Factor 1/8)	157	1960

Table 4.5: *Embedded memory requirements of 2-left and single hashing classifiers for simulated systems at varying load factors*

with that numerically predicted. This validates the starting assumption that for the systems considered, bins of load three make a negligible contribution to the total embedded memory cost of the system.

As expected, variance in the embedded memory cost when the system reaches steady state is observed. Again, at more favourable load factors, the relative frequency distribution characterising this variance is notably sharper and taller - one could speculate that for systems of infinite size this distribution would be a line of height 1, centred on the numerically predicted steady state value. For systems of practical size with appropriately chosen load factor, it appears (at least from this small subset of simulations) that the additional embedded memory required to accommodate variance and guarantee *overflow sufficiency* as $t \rightarrow \infty$ will be reasonable; an assumption which will be tested later in comparing a prototype hardware implementation with existing published work. Ultimately, for commercial systems development more rigorous characterisation would be required here, using systems of millions of items and larger load factors over long measurement intervals - such rigorous qualification and verification effort is considered beyond the scope of the current discussion. Notwithstanding this need for more rigorous characterisation, it is clear that *2-left* outperforms single hashing as a dynamic classification paradigm.

4.6.10 Defining Dynamic Systems Analytically

Noting that the numerical and system simulations discussed so far exhibit very favourable steady-state behaviour with well bounded embedded memory resource requirements, it is natural to inquire if one might establish a more general analytical model describing the system, such that firm conclusions could be drawn about the system behaviour as $t \rightarrow \infty$. In short, might it be possible to prove analytically that the system is unconditionally stable? Whilst it is possible to derive an equation describing the observed system steady-state, it appears difficult (if not intractable) to obtain a general solution from which any conclusions might be drawn.

To illustrate the complexities of any analytical approach, recall Equation 4.9 which described the dynamic system previously defined and modelled. One may separate this equation (which, for convenience, originally used the combined $x_i(t)$ notation) into two equations in $y_i(t)$ (reflecting the fraction of bins with load *at least* i in the left hand table) and $z_i(t)$ (reflecting the fraction of bins with load *at least* i in the right-hand table); M again represents the total number of items in the system:

$$\frac{dy_i(t)}{dt} = 2(y_{i-1}(t) - y_i(t))(2z_{i-1}(t)) - \frac{i(y_i(t) - y_{i+1}(t))}{M} \tag{4.10}$$

$$\frac{dz_i(t)}{dt} = 2(z_{i-1}(t) - z_i(t))(2y_i(t)) - \frac{i(z_i(t) - z_{i+1}(t))}{M} \tag{4.11}$$

The embedded memory cost analysis is based on the fraction of bins with load *exactly* i, given by $w_i(t)$ as follows:

$$w_i(t) = (y_i(t) - y_{i+1}(t)) + (z_i(t) - z_{i+1}(t)) \tag{4.12}$$

Differentiating, one obtains:

$$\frac{dw_i(t)}{dt} = \left\{ \frac{dy_i(t)}{dt} - \frac{dy_{i+1}(t))}{dt} \right\} + \left\{ \frac{dz_i(t)}{dt} - \frac{dz_{i+1}(t)}{dt} \right\} \tag{4.13}$$

Substituting Equation 4.10 and Equation 4.11 into Equation 4.13 and considering some stationary (in time) point where apparent steady state behaviour of the system occurs, one obtains, after some algebraic manipulation:

$$\frac{dw_i(t)}{dt} = \Delta^2(zy)_{i-1} + \Delta(i(\Delta(y_i(t)) + \Delta(z_i(t)))) = 0 \tag{4.14}$$

Where Δ and Δ^2 are the first and second difference operators [121, 122] with respect to the loading space i. The non-linearity introduced by the zy product term means a general solution to Equation 4.14 may be difficult or even impossible to find (since no standard differential solution forms are applicable here). Unfortunately then, at least by following the preceding analysis, it does not appear feasible to draw conclusions about the unconditional stability of the system. Given these difficulties, an alternative approach is to verify the system empirically, by considering more general operational models. In other words, to establish how stable and robust the proposed system is, let us try to break it.

4.6.11 More General System Use-Cases

Up to now consideration has been given to a system which behaves in a way which is easy to generalise and thus to model with a differential equation. That is, a dynamic classifier which is empty at $t = 0$, is filled to its capacity according to a *d-left* allocation of items governed by Equation 4.4, and then continues operation according to a "one-out-one-in" protocol at this system limit, governed by Equation 4.9 as $t \to \infty$. In real classification systems, operation is unlikely to be so rigorously defined. For example, in the context of flow monitoring, one may wish to delete a flow for which no packets have been seen in a certain time interval. Items in the associated classifier will thus typically be time-stamped and deleted according to some ageing protocol, which introduces the possibility of *bulk* deletions and insertions in the system when multiple items expire at the same time.

Since equations governing the insertion of items, and the deletion/insertion of items at a fixed capacity have already been established, to consider more general use cases one need only add an equation governing deletion in isolation. This follows naturally from Equation 4.9, where the insertion term may be removed such that the fraction of bins with load at least i under bulk deletion is given by:

$$\frac{dx_i}{dt} = -\frac{\left\{Floor\left\{\frac{i}{2}\right\}\right\}(x_i - x_{i+2})}{M'} \tag{4.15}$$

Note that M is replaced with M' in Equation 4.15 since under continual deletion the number of items in the system is no longer static, and must be accounted for in any numerical approximation. With three modes of operation (insertion, deletion, both) more general system use-cases may now be modelled, though one assertion remains in place. That regardless of how items arrive or expire, the system has a fixed capacity M.

A system with 16384 items and 65536 hash bins (load factor $\frac{1}{4}$) was chosen for further analysis. In the first use-cases considered, items are allocated up to the system capacity, and deletion/insertion continues at the system limit (as before) until steady-state behaviour is observed. One then perturbs the system by deleting and reinserting items in bulk, again up to the system capacity, whereupon deletion and insertion at the system limit recommences. Figure 4.27 shows the results for both numerical approximation and system simulation of this behaviour for bulk deletion and insertion of 2000 items (subfigures (a) and (b)) and 8000 items (subfigures (c) and (d)). Excellent correlation is again observed between the numerical model and the system simulation, and after perturbation the system reaches the same steady-state previously observed. The number of embedded memory locations required never exceeds this value (associated with deletion and insertion at the system limit) - approximately 408 for the system considered.

To build confidence in the system, some further general use-cases were modelled. Since tailoring the software simulation for the systems of Figure 4.27 proved time consuming, and since consistent correlation with the numerical results had been established, it was decided to use only numerical models for the remainder of the use-case analysis. Four additional scenarios were considered. In the first, system capacity was reached by continuously inserting 2 items and deleting 1. In the second, upon reaching the steady state at the system capacity, an additional 8000 items were added and deleted in bulk, followed by continual deletion and insertion at the system limit - note that this use-case is actually illegal, since the system limit is exceeded, but is included as an interesting perturbation of the steady-state nonetheless. In the third, the 16384 items are inserted in two batches, with interim insertion/deletion phases. That is, 8192 items are inserted, one then deletes and inserts items such that the number of items remains at 8192, one then adds a further 8192 items, and recommence deletion and reinsertion at the system limit. In

the final scenario, the system limit is reached by staggered insertion of 6000 items followed by deletion of 1000 up to 15000, with a final top-up insertion to 16384, followed by deletion and insertion at the system limit. The results are shown in Figure 4.28.

One may observe that, with the exception of the use-case where the system limit is deliberately exceeded, the embedded memory location cost never exceeds the level predicted by deleting and inserting items continuously at the system limit. Even for the illegal use-case, the same steady-sate value is reached when the illegal items are removed. Thus, again with the caveat that more rigorous characterisation would be required as part of commercial systems development, it is asserted that *2-left* is robust and stable under dynamic operation.

4.6.12 Simulations with Real IPv4 Data

As an additional check on the viability of the proposed implementation, real IPv4 traces including some from the National Library for Applied Network Research [123] were used as the input to a system simulation with 16384 items and 65536 hash bins. As before, to establish the apparent upper bound on the embedded memory requirements, items are allocated by *2-left*, with items subsequently deleted and inserted at the system limit. The traffic details are summarised as follows:

- Figure 4.29(a): *Random input data* - As previous simulations.
- Figure 4.29(b): *Bell Labs Input Trace* - A one week contiguous internet access IP header trace collected at Bell Labs research, Murray Hill, New Jersey.
- Figure 4.29(c): *Leipzig II Trace* - An illustrated 1-day GPS-synchronized IP header trace captured simultaneously at either side of the University of Leipzig's central Internet access router.
- Figure 4.29(d): *Salzburg FH Trace* - A contiguous internet access IP header trace collected at the University of Applied Sciences (Fachhochschule) Salzburg's packet capture point.

The results are shown in Figure 4.29 and suggest that repeatable and consistent behaviour with real IPv4 data as inputs to the system is achievable. The highest embedded memory cost (observed when using the Bell Labs traffic) was approximately 475, compared with a maximum

value of approximately 450 when using random input data. In general, the correlation between simulations using synthetically generated random keys and those using real data is good. Use of better hash functions, or additional random multipliers as suggested in [16] should improve repeatability across such simulations further.

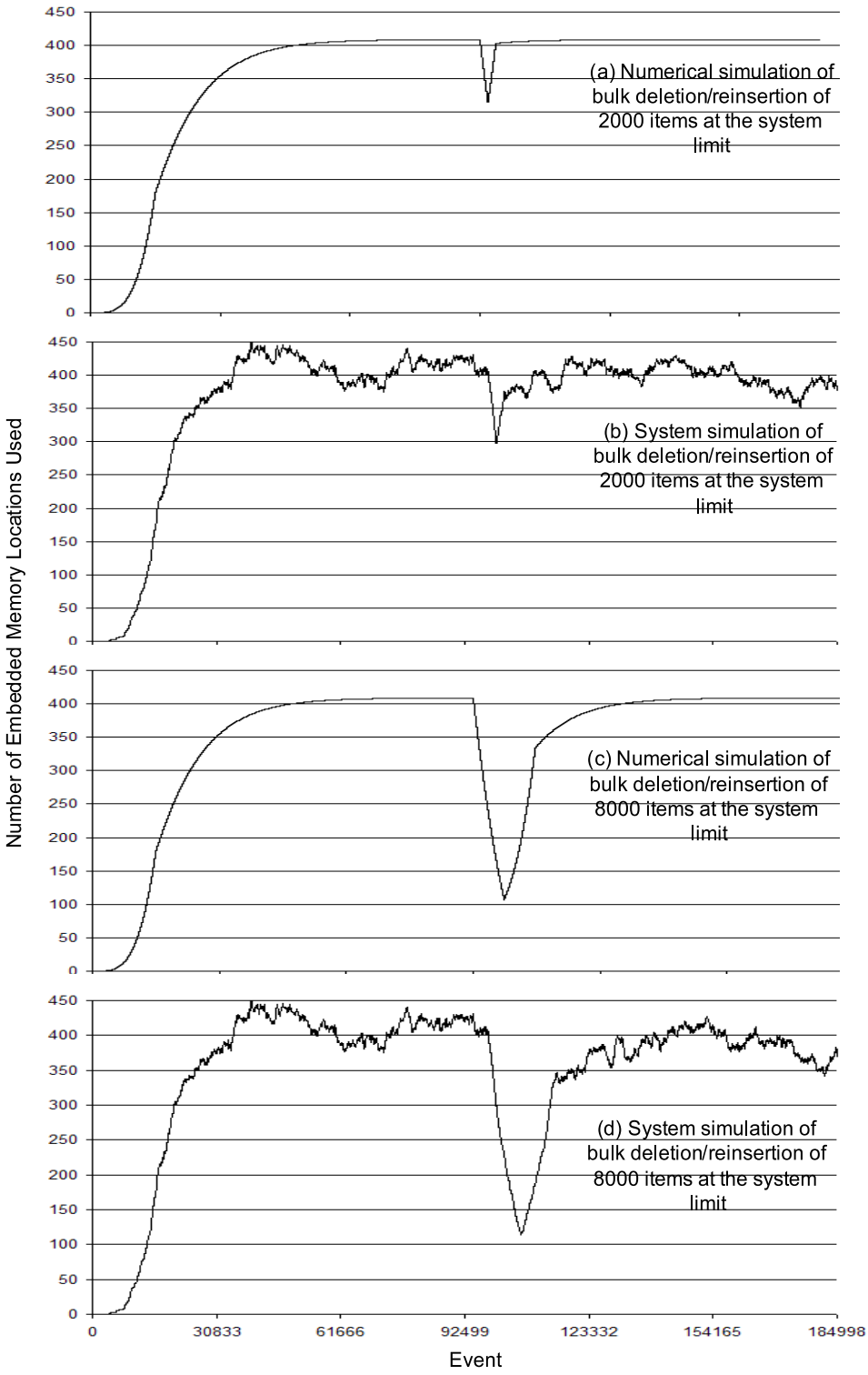

Figure 4.27: *Embedded memory utilisation for a 2-left classifier with 16384 items and 65536 bins, showing bulk deletion and insertion of items*

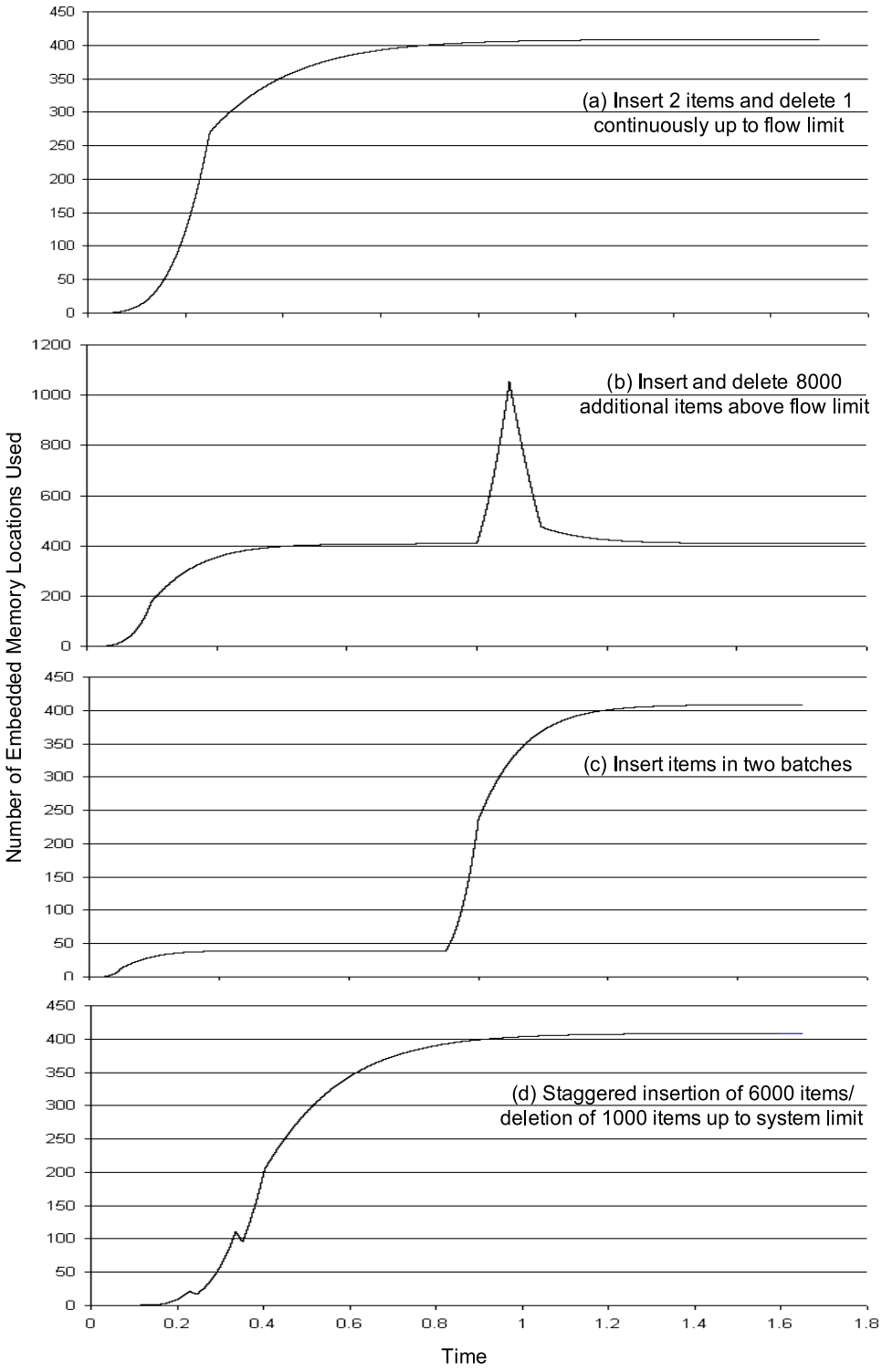

Figure 4.28: *Embedded memory utilisation for a 2-left classifier with 16384 items and 65536 bins, under more general use-cases*

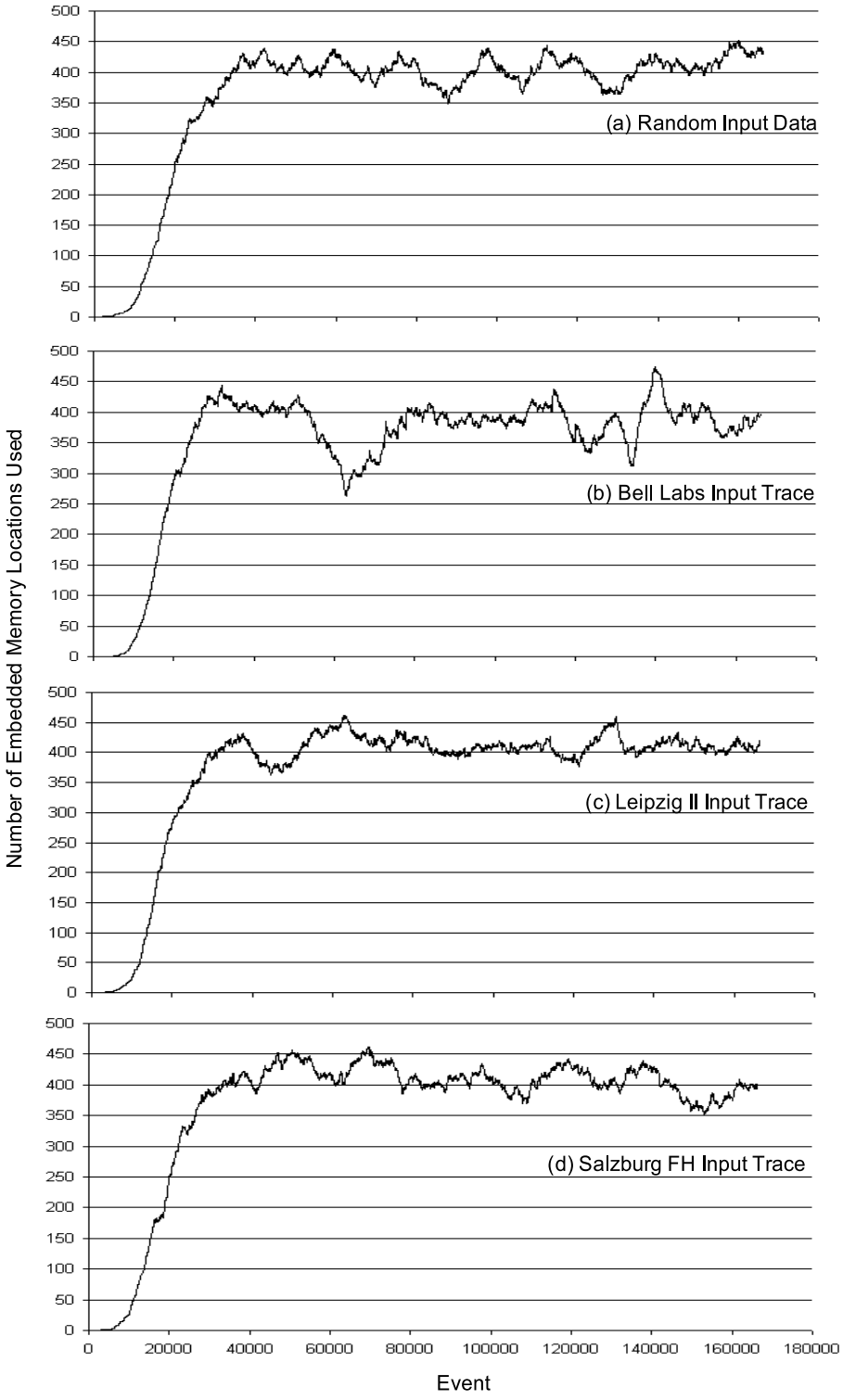

Figure 4.29: *Embedded memory utilisation for a 2-left classifier with 16384 items and 65536 bins, with IPv4 trace data as input*

4.7 A Prototype Hardware Implementation

The preceding analysis has produced some interesting and potentially very useful outcomes. The results obtained by considering dynamic systems, initially following a simple rule whereby new items are added on a "one-out-one-in" basis and latterly considering more general modes of operation, suggest that an FPGA implementation based on *2-left* is viable and robust. Further, it appears that at appropriate load factors, such an implementation will perform dramatically better than single hashing. On this basis, and to facilitate comparison with other contemporary classification approaches, it was decided to proceed to the investigation and implementation of an FPGA-based hardware prototype. Recall the solution proposed in 4.6.5. Items will be allocated according to a *2-left* insertion into two hash memories (nominally implemented in external RAM). If the RAM location chosen by *2-left* is free, the item will be stored in external memory at that address. In the event of a collision, the item will be stored in FPGA embedded memory. Upon an item query, the external and internal memory spaces will be searched in parallel.

To facilitate faster prototyping it was decided that, rather than develop a full discrete memory system, the prototype be implemented exclusively in embedded FPGA memory. Partitioning the prototype into areas of block RAM emulating the "external" hash tables (which would ultimately be implemented in discrete SRAM or DRAM) and a separate area of block RAM fulfilling the collision resolution function, was deemed appropriate to establish proof-of-concept. Modelling the system in this way removes the complexity of implementing embedded memory controllers (a non-trivial task, especially in the DDR-DRAM case), and allows many more classification decisions in a given test interval, since embedded RAM offers higher bandwidth access. Note that the following discussion will still refer to "external" hash tables (despite implementing the structures internally in this instance) since the hash table locations in the prototype will behave exactly like their discrete equivalents.

Embedded FPGA memory is, of course, a finite resource. The prototype system size must therefore be chosen appropriately such that the combined requirements of the two *d-left* hash tables and the corresponding *overflow* memory do not exceed the embedded RAM capacity of the chosen FPGA.

The required capacity for any given system can be estimated by the numerical means already detailed. However in considering a dynamic system where the hash memory and collision

resolution memory are physically separate, one must extend the numerical analysis by introducing the concept of item *promotion* - a prerequisite to maintaining memory efficiency in the proposed implementation.

4.7.1 Promotion and Memory Efficiency

Consider the simple hash tables shown in Figure 4.30, following the balls into bins analogy. Subfigure (a) shows a generic (implementation independent) hash table, where the bins have theoretically infinite capacity. Thus, for example, although items 1,8 and 11 are all allocated to location A1, there is sufficient capacity to store them all. Every item maintains a permanent association with the hash bin it is allocated to. The deletion of items is as straightforward as insertion. If say, item 5 were deleted, the load in bin A7 would reduce from 2 to 1, with item 10 still present in that bin.

Finite hash bin capacity complicates matters. In the proposed implementation, one may only store a single item at each address in the external hash tables. In Subfigure (b), since items 8,9,10 and 11 hash into addresses which are already occupied, they are stored in embedded FPGA memory for collision resolution, and any associativity with a hash table address is lost. In this case, when item 5 is deleted from address A7, a memory inefficiency is created. Location A7 is now free, and item 10 hashes to that location, but is not using it. Since, in this embodiment, any associativity between item 10 and location A7 was lost when the former was stored in collision resolution memory, there is no way to *promote* the item back into external memory and fill the “memory hole”. Thus, although location A7 now has a nominal load of 1, the item comprising that load is still stored on-chip, and the embedded memory cost of the system is higher than it should be.

4.7.1.1 Modelling Memory Inefficiency in 2-left

The discussion in 4.7.1 raises concerns about the impact of inefficient memory utilisation in dynamic systems as $t \rightarrow \infty$, particularly with respect to the steady-state behaviour previously observed. It was therefore deemed necessary to develop a better understanding of this inefficiency, and update the numerical models for dynamic systems to reflect this. Again, to simplify matters, let us assume that a favourable load factor is chosen and thus that bins of load 3 and higher have negligible impact on the embedded memory costs, and can be ignored.

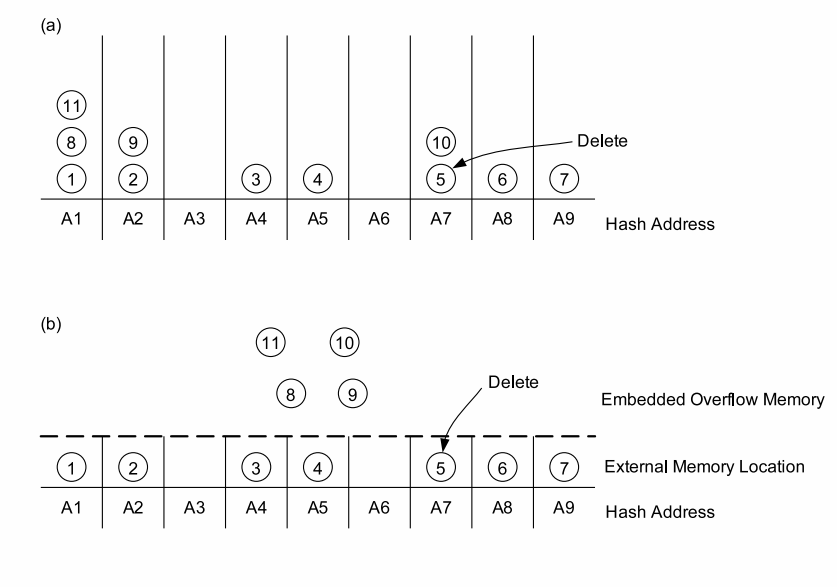

Figure 4.30: *Simple hash tables - implementation independent (a), and with physical separation between primary hash space and collision resolution overflow (b)*

One may thus define a hash bin to be *inefficient* only if it has a load of 1, with the item stored in the embedded FPGA overflow memory rather than in the hash table location directly.

So how might a hash bin become inefficient? By the definition above, one must find a hash bin with load 2, and delete the item in that hash bin which is physically stored at the external hash location. This results in a bin of load one, with an associated item stored in the overflow memory. Conversely, an inefficient hash bin may be returned to an efficient state if the item persisting in overflow memory is itself deleted, or if a future insertion "plugs" the memory hole. All three possibilities must be accounted for in mathematically modelling memory inefficiency.

Consider initially the left-hand hash table in a dynamic *2-left* system. Let $p_1(t)$ be the fraction of inefficient bins in the left-hand table - that is, the fraction of bins in the left-hand table, with load 1, where the item is stored in overflow memory. As for all the preceding dynamic analysis, let dt be the interval of time during which an item is deleted and a new item inserted. For $p_1(t)$ to increase in this interval, one must choose an item for deletion which is in a hash bin of load 2, and is physically stored in the external hash table location - let us call this "Event 1". With a uniform random deletion distribution, the probability of Event 1 occurring is given by:

$$P(E_1) = \frac{2l_2(t)N}{2M} = \frac{l_2(t)N}{M} \tag{4.16}$$

where $l_2(t)$ is the fraction of bins in the left-hand table with load *exactly* 2, N is the number of bins in the system and M is the number of items. For $p_1(t)$ to decrease in the interval dt, one must either delete an item from an inefficient bin - let us call this "Event 2", or insert an item into an inefficient bin - let us call this "Event 3". Event 2 and Event 3 have probabilities given by Equations 4.17 and 4.18 respectively:

$$P(E_2) = \frac{p_1(t)N}{M} \tag{4.17}$$

$$P(E_3) = 2x_3(t)p_1(t) \tag{4.18}$$

The net change in $p_1(t)$ is thus a simple combination of these 3 probabilities, given by Equation 4.19, and by completely analogous arguments an equivalent differential equation for the fraction of inefficient bins in the right hand table, $q_i(t)$, is given by Equation 4.20, where $r_2(t)$ is the fraction of bins in the right-hand table with load *exactly* 2.

$$\frac{dp_1}{dt} = \frac{l_2(t)N}{M} - \left\{ \frac{p_1(t)N}{M} + 2x_3(t)p_1(t) \right\} \tag{4.19}$$

$$\frac{dq_1}{dt} = \frac{r_2(t)N}{M} - \left\{ \frac{q_1(t)N}{M} + 2x_4(t)q_1(t) \right\} \tag{4.20}$$

The impact of inefficient memory utilisation may then be modelled numerically. At time $t = t_s$ the static allocation phase has just completed, and since there have been no deletions, $p1(t) = q1(t) = 0$. From $t_s \leq t < t_t$ where t_t is again some arbitrary but finite simulation time one may then model Equations 4.19 and 4.20 in parallel with the original dynamic system model to establish how many bins of load 1 are inefficient at the end of the process. After completing this process for load factors of $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$ and $\frac{1}{16}$ it was observed that the system continued to exhibit steady state behaviour in the dynamic phase, but that the embedded memory cost at which this steady state was reached was approximately double that for system without physical separation of hash and overflow memory space. A new set of software simulations were also constructed for each of these load factors. The results are shown in Figure 4.31, with good correlation again observed between the actual system behaviour and the numerically predicted steady state value, shown as a dashed line in each subfigure.

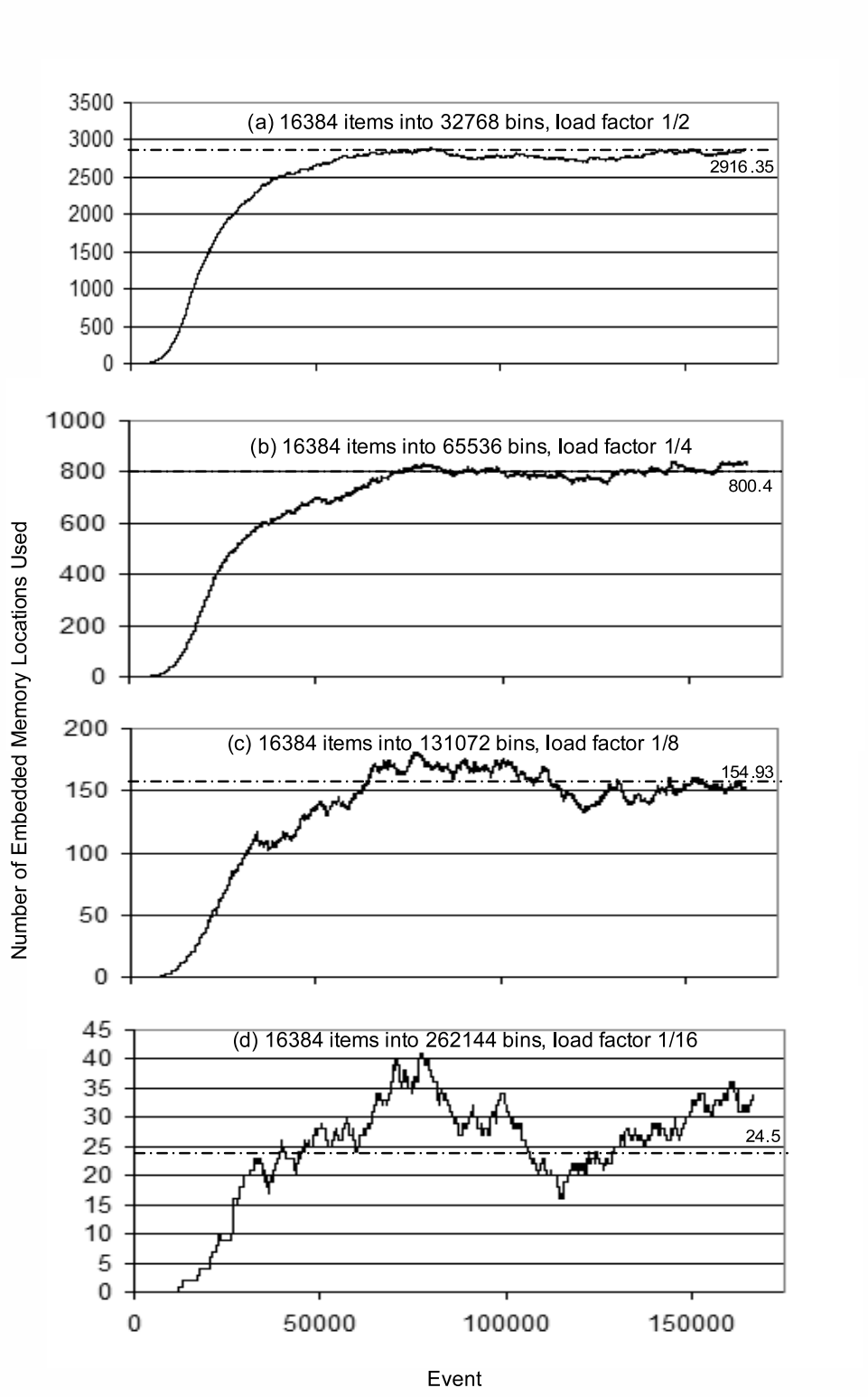

Figure 4.31: *Embedded memory utilisation for a 2-left classifier with 16384 items and variable bins, under dynamic deletion and insertion of items without promotion*

4.7.1.2 Design Trade-Offs in Implementing Promotion

One need not live with the memory inefficiencies just described, but one cannot resolve the problem without some cost in terms of logic utilisation and performance. The problem lies in the loss of associativity between the item stored in the embedded FPGA overflow memory and the external table location to which it initially hashed. To resolve this, in addition to storing the item itself in embedded FPGA memory, one could store two additional pieces of information - a flag to indicate whether the item was allocated to the left or right hash table, and the address in that table at which the item collided, causing it to enter the overflow memory in the first place. Thus, when an item is deleted, one could search the internal overflow memory for entries with a matching flag/hash pair, and select the item for promotion back to the external memory location. This process restores full associativity between overflow items and hash table addresses, removing the inefficiency detailed in 4.7.1.1.

Clearly there is a logic cost. Say the items to be classified are IPv4 keys, 104 bits long and are being allocated into a hash space of $2^{20} = 1,048,576$ locations. To support the associativity just described one must store the key itself plus an additional 20 hash address bits and a left/right flag bit - a 20% overhead on every item stored in the overflow memory. There is also a cost in terms of update performance, since in addition to the first parallel search to look for the item to be deleted, one must initiate a secondary search of the internal memory to look for promotion candidates.

Having given due consideration to all of the above points, it was decided to implement a system *without* any promotion mechanism, since initial comparisons with other published approaches suggested that even when running at sub-optimal efficiency, the proposed implementation offers performance improvements. This point will be revisited in more detail later.

4.7.2 Prototype System Dimensions

Recalling the decision to implement a classifier entirely in embedded FPGA memory, the initial prototype system was specified with the following parameters:

- 32768 32-bit[6] hash table locations, comprising *left* and *right* tables of 16384 locations each.

[6] 32 bits are not actually required here, since the input items are accessed via pointers. 32 bits allocated to provide headroom for any later changes or experimentation.

- An maximum input capacity of 4096 32-bit items, with an additional 32 bits allocated to store associated state or context information.
- A collision resolution capacity of 128 items.

The hash table locations can be implemented in embedded block RAM. The FPGA used in the prototype platform discussed in 2.3.4 has 216 such RAMs available, each providing 512 x 36-bit memory locations. The hash space thus requires 64 of these. Additionally one requires 16 block RAMs to support a system *state table* which will keep a list of the current contents and any associated information or actions. In real world applications, such a table might be used to store timestamp information, or to associate some filtering or forwarding decision with each input. The collision resolution space can be implemented in distributed RAM, utilising around only 1% of the available logic resource to support 128 x 60 bit entries. A small amount of additional block RAM resource is also required to control allocation and deallocation of state memory, and to support diagnostics.

4.7.3 Basic Circuit Operation

Recall once again the basic operation of the *2-left* algorithm. When an item arrives to be allocated, one choses a random location in the left-hand table, and a random location in the right-hand table. The item is placed in the location with the lowest load, and ties are always broken by placing the item in the left hand table. Mirroring this behaviour, and maintaining the consistency of the *2-left* algorithm with physical separation of memory areas, a proposed packet classification architecture is shown in Figure 4.32. A description of basic circuit operation follows.

4.7.3.1 Allocating a New Item

Assume that the current input item does not already exist in the system. When the new input item arrives, two corresponding hash functions for that item are computed. Function $H1$ provides a pseudo-random address into the left hash table, and function $H2$ a pseudo-random address into the right. The left and right hash table locations each return a *pointer* into state memory, where the input items of interest are actually stored, together with any associated context information. Indirect addressing in this fashion improves the efficiency of the implementation. Since the items themselves must be stored in state memory anyway, there is

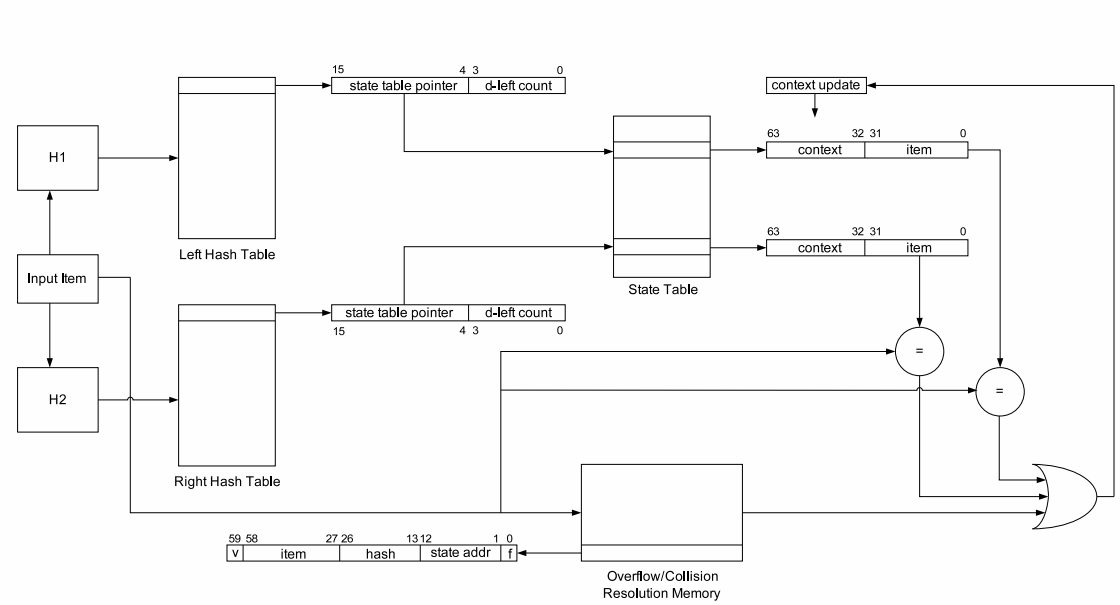

Figure 4.32: *Initial prototype classification architecture*

no requirement to explicitly store them in hash memory - a pointer from hash memory into state memory suffices. This decouples the size of the hash memory from the input item length. In the case of the initial prototype, since the system has a capacity of 4096 items, each hash location need only store a 12-bit pointer (and a count, discussed in more detail later).

The pointers returned from hash memory are then used to address the state memory which returns the items at the addressed locations for comparison with the current input. At the same time, a fast parallel search of the overflow memory compares all of its contents with the current input. Since the item does not already exist in the system, all three comparators (left hash table, right hash table and overflow memory) will return a miss, so the current input may be identified as new. At this point, the *2-left* counts associated with each of the hash table locations are examined. If the right hand count is less than the left the item is associated with the right hand table, otherwise the item is associated with the left, and the corresponding count at the hash table location incremented.

The term *associated* is used deliberately since at this point, no decision has yet been taken as to where the current item will *actually* be stored. If the item is allocated left, and the left hand hash table location is free, one writes a pointer into state memory to the left hash table, and writes the item itself into state memory. If the item is allocated left, but the left hand hash table is

already occupied by a pointer to another item - a *collision* - then the item itself is written to the overflow memory, along with its hash table address, a flag bit to indicate that the item collided in the left table, and the item's subsequent address in state memory. Analogous arguments apply to collisions in the right hash table. A *valid* bit is set in the new overflow memory location, to indicate that it is a valid candidate for future searches.

4.7.3.2 Updating an Existing Item

If the current input already exists in the system, one of the comparators will return a hit. If the item is found in one of the hash memories, the pointer into state memory at the appropriate location is used to update the context information associated with the input. In the initial prototype, one simply increments a context count associated with the input. Note that this is *not* the count associated with *2-left*, merely the number of times one has seen an input in the current classification period. If the item is found during the parallel lookup of the overflow memory, the corresponding state table address stored at the overflow location is used in similar fashion to update the context.

4.7.3.3 Deleting an Old Item

In practical classification applications, such as TCP flow monitoring and packet filtering, one may wish to stop monitoring a particular item and remove it from the system, perhaps based on some ageing protocol, whereby inputs which have not been seen over a designated monitoring period are discarded. Effectively, somewhere in the software monitoring hierarchy, a decision is taken to remove an item from the state memory. This item is passed to the classification engine along with a delete flag, and traverses the same datapath as before.

The item for deletion is hashed to produce two addresses into hash memory, which return pointers into state memory as before. In parallel, the overflow memory is searched. Since the item has been flagged for deletion, one of the comparators will return a hit. If the item is found in hash memory, the associated *2-left* count is decremented, and the pointer to state memory cleared. The state table location is subsequently freed for use by any new input item which arrives.

If the item for deletion is found in the overflow memory, the hash address and flag stored at that overflow location are used to decrement the appropriate *2-left* count in hash memory. The state

table address stored at the overflow location is subsequently freed for use by any new input item which arrives, and the *valid* bit stored at the overflow location is cleared to discard it from any future searches of the overflow memory.

4.7.4 Implementing Fast Overflow Search

The ability to search the overflow memory efficiently is key to the performance of the overall engine. For a system with hash space implemented in DDR-DRAM one would seek to complete the on-chip embedded memory lookup in around the same time as it take to retrieve a state table pointer from the discrete memory device. Completing the on-chip search more slowly than the DRAM access time means that this becomes the performance bottleneck in the system; completing the on-chip search significantly faster than the DRAM access time is redundant, since one must always wait for the state table pointer to complete the classification operation. A full analysis of discrete hash and state memory access timing is dependent on the memory device type, hash address width and memory data width and was deemed beyond the scope of the current implementation. Instead, based on similar parallel lookup schemes implemented previously by Aliathon, it was decided initially to allow 4 system clock cycles per internal item lookup.

4.7.4.1 A First Attempt

The prototype system allocates 4096 items into 32768 locations; a load factor of $\frac{1}{8}$. Numerical analysis for an ideal system with this load factor predicts that under dynamic operation, a maximum of approximately 0.0006 x $32768 = 19.66$ items will end up in overflow memory. Since the initial prototype does not implement any promotion from overflow memory, one must also account for memory inefficiency as modelled by equations 4.19 and 4.20, which predict that this will increase to 0.001182 x $32768 = 38.73$ items. An overflow memory space of 128 items was thus chosen to give more than enough headroom to characterise the variance in this numerically predicted loading. To allow an input comparison every 4 system clock cycles, the overflow memory initially proposed was thus structured as shown in Figure 4.33.

Since in any given block RAM one may only address a single memory location in each clock cycle, the overflow memory is constructed of 8 block RAMs of 16 32-bit locations each. This supports storage of up to 128 32-bit inputs. A common read pointer steps through each RAM,

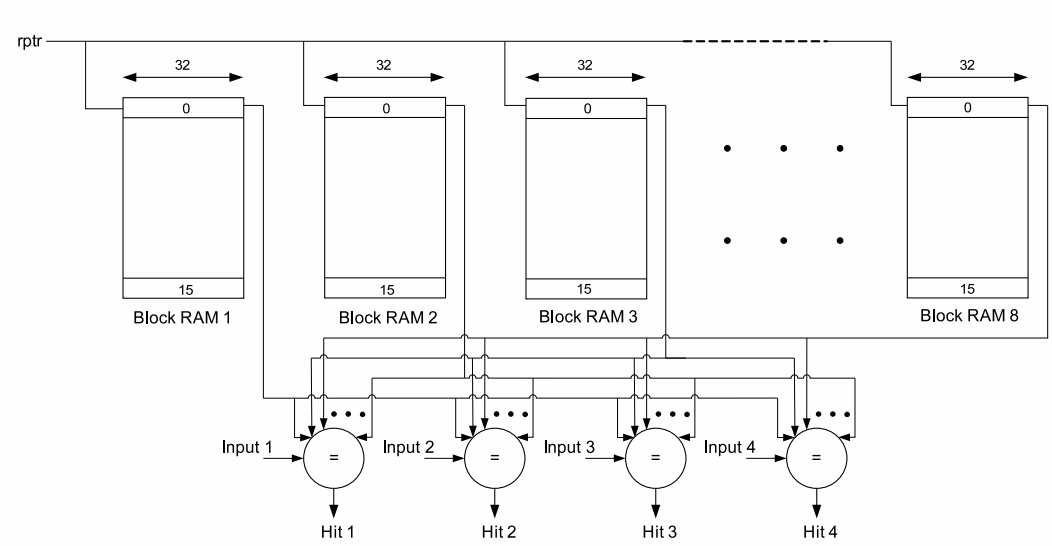

Figure 4.33: *Block RAM architecture to facilitate an input comparison every 4 system clock cycles*

incrementing once every clock cycle such that in 16 clock cycles, the entire contents of the overflow memory may be read. Since one comparison every 4 clock cycles is required, 4 inputs are buffered and compared in parallel during each traversal through the overflow memory. Thus, in 16 clock cycles, 4 inputs may been compared with every overflow memory location - equivalent to an average throughput of 1 comparison every 4 clock cycles.

4.7.4.2 Complexities and Limitations

Initial implementation efforts based on the overflow memory structure proposed above raised a number of difficulties, both in the context of the current prototype and in the context of scalability to systems of practical size. By way of illustration, consider the simplified 4-bit comparator logic shown in Figure 4.34, for comparison of a 4-bit number a with 4 peers b, c, d, e. To compare a with one of its 4-bit peers - b say - one needs 8 logic inputs which, in FPGA terms, utilise 2 4-input LUTs . The first LUT compares a_0 with b_0 and a_1 with b_1, the second compares a_2 with b_2 and a_3 with b_3. The results of the 4 individual bit comparisons are then aggregated in a second layer of logic. To aggregate all the parallel comparisons (with b, c, d and e), a third layer of logic is required.

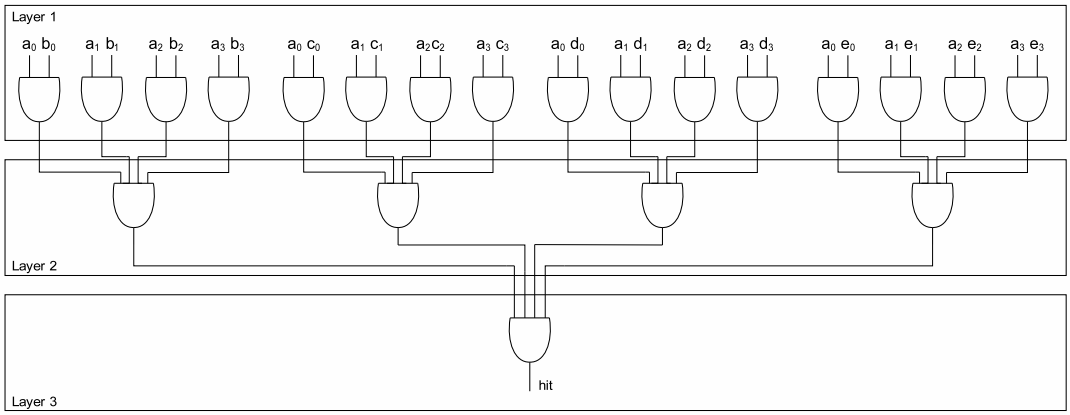

Figure 4.34: *Parallel comparison of a 4-bit number with 4 others*

In the case of the initial prototype, 32-bit numbers are compared. Thus a single comparison requires 64 inputs to the first layer of logic, and produces 16 outputs to a second layer, which in turn produces 4 outputs to a third layer, which produces the result. A fourth layer of logic is required to aggregate the results of the four parallel comparators. Whilst this basic comparator logic may readily be made to run at speeds approaching 200MHz on the target platform for the classification prototype, it raises concerns about scalability to IPv4 and particularly IPv6 systems. Comparison of 2 128-bit input keys, for example, presents 256 bits to the first layer of logic. Aggregation of 4 parallel comparisons of this width requires 5 layers of logic, and significant FPGA routing resource, both of which act to make timing closure more difficult.

A further difficulty arises as a direct consequence of the latency through the system. In the case where four parallel comparisons are performed, 16 clock cycles are required to buffer 4 inputs and read the appropriate pointers and context information from memory, 16 clock cycles are required to step through the overflow memory for comparison, and 16 clock cycles are required to write updated information to the memories based on the results. The non-zero time (or latency) between reading context information from state memory and writing updated context information back represents a non-atomic read-modify-write loop, which introduces the potential for spurious context update and degraded system capacity.

To illustrate the problem, let us consider some specific scenarios which are likely to occur in real classification systems. Say the latency through the classifier is t_l, and an item - a packet header A - already exists in state memory with an associated count of 6. That is, packets with

header A have been seen by the system 6 times previously during the current classification period. There then follows a burst of 4 input packets with header A during a time interval less than t_l. The first of these packets to arrive will trigger a match with A in the state table, and cause the associated count to be incremented. However, the latency in the system is such that the subsequent inputs in the current burst, fetch a count from state memory which has not yet been updated. Thus each of the 4 inputs fetch a count of 6 and increment it to 7. As a result, after the final packet in the burst is processed the count in state memory associated with A is only 7, when it should be 10.

As another example, let us assume that packet header A does not currently exist in the system. There then follows a burst of 4 input packets with header A, again in a time interval less than t_l. The first packet to arrive will generate a miss in all the system comparators, and cause a new state table entry to be generated for A. However, the latency in the system is such that the subsequent inputs in the current burst also generate a miss in all the system comparators (since the new entry for A has had insufficient time to be created), and three new entries for the remaining packets are also created, causing a spurious reduction in the system's capacity. Similar scenarios might be envisaged for items scheduled for deletion, although since the deletion mechanism is controlled directly by the system, this could be engineered to avoid any contention.

To preserve the integrity of context updates, one could introduce a mutual exclusion scheme [124] whereby context information may only be accessed under semaphore control, and is thus always guaranteed to be valid. Such a scheme introduces non-determinacy in state table access time however - an unacceptable characteristic, considering the initial design motivations. A better solution in the context of the current implementation is to introduce a cache, which restores the atomicity of system context update using local memory access. The basic idea for a system with 4 inputs buffered at any one time, and a read-modify-write latency of 16 clock cycles is illustrated in Figure 4.35.

The cache contains the item and context information for inputs which have been classified but not yet updated in state memory. It is positioned after the primary classification logic, acting like a shift-register, with inputs from the primary classification logic and outputs used directly to update the state table. As shown in Figure 4.35, items A, B, C and D already exist in the cache. That is at some time $t - x$ where $x \leq 16$, A, B, C and D have already been seen by the system. Note that the other cache locations are also occupied, but are marked as "don't cares"

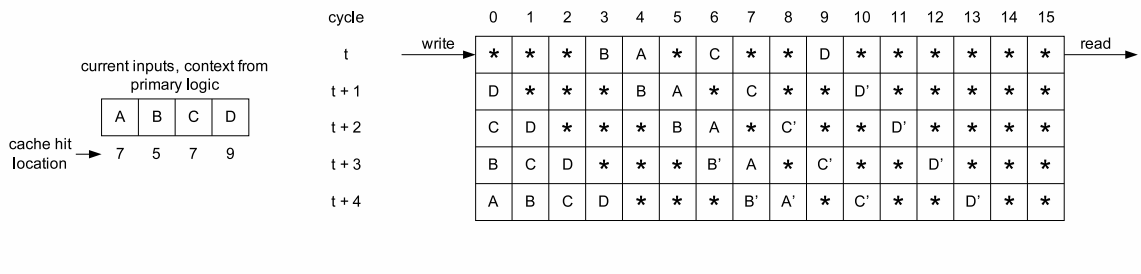

Figure 4.35: *Cache structures to preserve context accuracy*

since they have no bearing on the outcome of this example. The items A, B, C and D arrive at the system again, and during cycle t have been processed by the primary classification logic, which fetched the associated (and now out-of-date) context information from the state table.

To check that the context information fetched from the state memory is valid, one checks to see if the current item already exists in the cache. During cycle t one searches for D in the cache. This search returns a hit at location 9. The context information originally fetched by the primary classification logic is thus deemed out-of-date, and overwritten based on the context information stored at location 9 in the cache. In the case of maintaining a simple count associated with D, if the count value returned by the primary classification logic was y, then the value stored in the cache at location 9 would have been $y+1$ and the value used to overwrite the new occurrence of D would be $y + 2$.

During cycle $t + 1$, the cache shifts right. Location 15 is read from the cache and written to state memory, item D and its new context are written to location 0, and the original version of D (now at location 10) is tagged as old (D') and not included in any further searches. Also in cycle $t + 1$, the search for the next input item C commences. This returns a hit at location 7 and the process repeats.

Unfortunately, comparator performance and latency in the proposed architecture are antagonistic problems. One may try to ease logic timing in the primary classification logic by pipelining or allowing additional clock cycles to traverse the overflow memory space. This increases the latency through the system however - necessitating a larger cache - and since a parallel search of the cache itself requires comparator logic, the timing problems are transposed rather than solved. Conversely, to reduce the latency through the system to the point where little or no cache is required would demand prohibitive amounts of parallelism

in the primary comparator. Acknowledging that these difficulties would be compounded in systems with long inputs and high packet rates, it was decided to abandon the approach of brute-force parallel search and completely restructure the overflow memory - again by taking advantage of the *d-left* algorithm.

4.7.4.3 A 2-level, d-left Architecture

From the preceding analysis it is evident that after allocating 4096 items into 32768 hash locations, one would expect a maximum of 38.73 items to end up in overflow memory during dynamic operation of the old system. Given the attendant difficulties of resolving these overflow items directly in embedded memory, it was decided to apply a different approach. Namely, rather than store overflow items from the external hash space directly, one allocates them into a small secondary *2-left* scheme, implemented in block RAM. Whilst this at first appears somewhat inefficient (since embedded block RAM is a scarce resource, and hash memories are sparsely utilised by design) it goes a long way to simplifying the resolution of entries which end up on-chip.

For the prototype system, approximately 40 items will end up colliding in the primary hash space. Allowing 512 locations in the second level *2-left* allocation thus provides a load factor of better than $\frac{1}{8}$. By an identical numerical approximation to that applied to the primary allocation, one would expect less than 0.0006 x $512 = 0.3072$ items to overflow both the first and second *2-left* allocations. This means that in theory a very small, register-based, third level overflow memory is now sufficient to guarantee robust dynamic operation, since only very rarely will items collide at both levels in the hash space. This removes the need for brute-force parallel lookup on all but the smallest structures in the design. The proposed architecture is illustrated in Figure 4.36.

4.7.5 New Circuit Operation

Allocation, update and deletion of items in the new system is similar to the previous embodiment, except that one now has a secondary hashing layer, which replaces the parallel block RAM lookup. One must be a little more careful in maintaining correct load counts here, to ensure that allocations at both levels of the *2-left* scheme are made correctly. Consider the case shown in Figure 4.37.

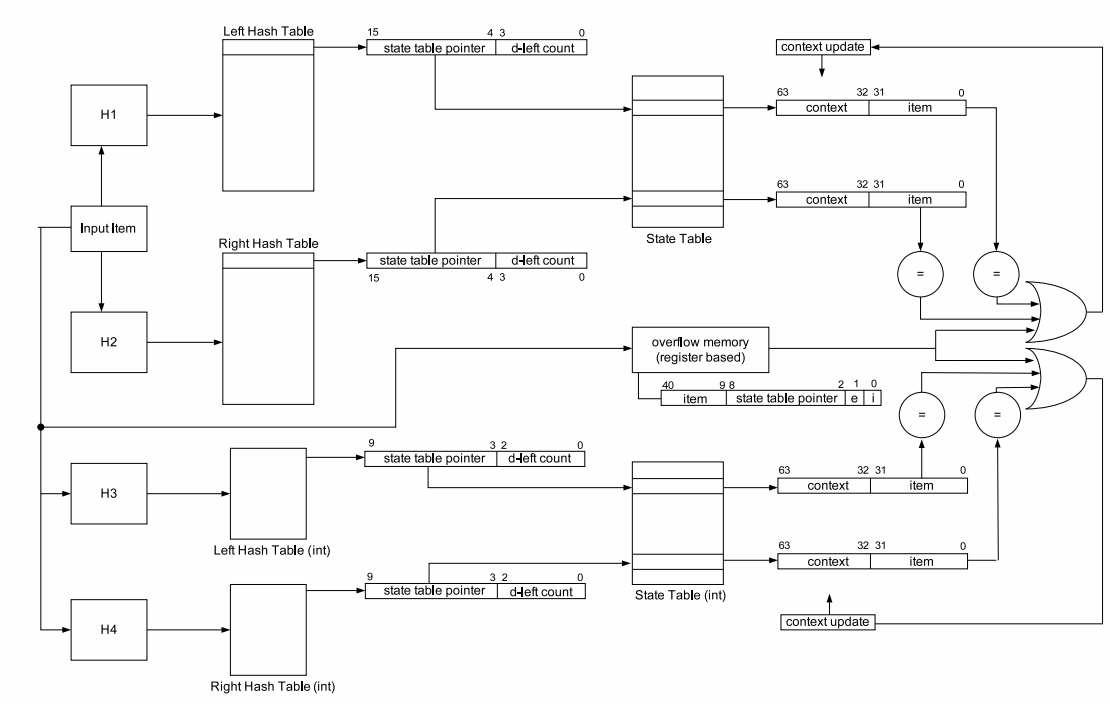

Figure 4.36: *Revised prototype architecture with 2-level, 2-left hashing*

A new input item C is allocated left by the primary *2-left* scheme, and collides with a pointer to an existing item A. It thus filters through to the secondary allocation, where it is placed right, but collides again at this level with a pointer to another existing item B. Item C must therefore be placed in the overflow memory, and since it hashed left in the primary allocation, and right in the secondary allocation, one must increment the appropriate counts at each level. In this way, the *2-left* protocol is adhered to at all levels in the system. When C is written to the overflow memory, it is tagged with an external and internal flag, such that in the event that C is deleted from the system at some point in the future, the appropriate counts may be decremented.

4.7.6 Resource Utilisation

Whilst in systems of practical size, the external hash space and state memory would be implemented in discrete DRAM and SRAM devices, for the initial proof of concept here, the entire infrastructure is implemented entirely in embedded FPGA memory. In the revised implementation, the first layer of the architecture remains unchanged. One initially allocates 4096 items into 32768 hash locations, the latter utilising 64 embedded block RAMs. As before,

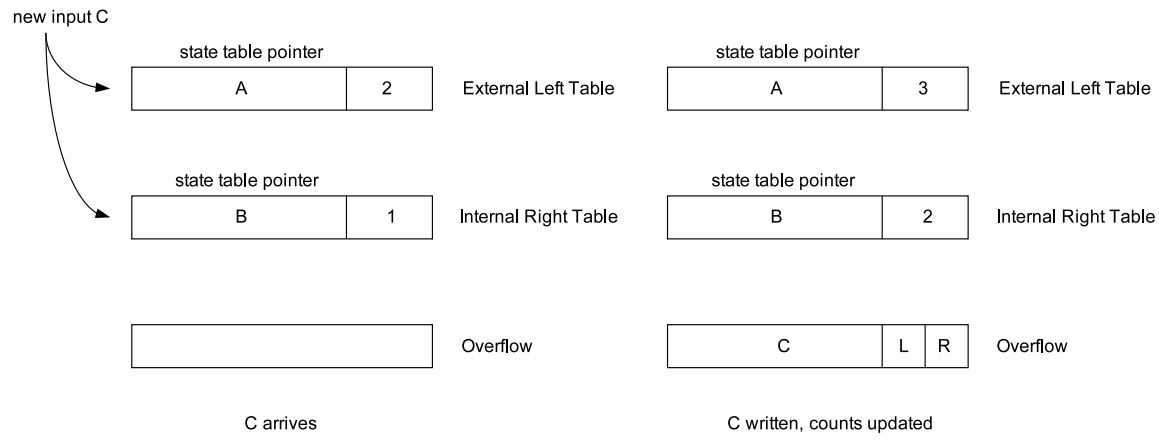

Figure 4.37: *Maintaining correct load counts in a 2-layer, 2-left allocation*

one also requires a further 16 block RAMs to support the *state table* associated with the entries in this first hashing layer.

Now, instead of implementing a contiguous overflow memory in distributed RAM, one constructs a secondary hashing layer in embedded block RAM. To achieve a load factor of better than $\frac{1}{8}$ at this level, one needs 512 hash locations - easily supported in this case with a single block RAM. An additional block RAM is required to support the second level *state table*, and approximately 328 bits of register-based logic are sufficient to provide a third level overflow capacity of 8 items. A small amount of additional resource is again required to control allocation and deallocation of state memory, and to support diagnostics.

4.7.7 Numerical, Software and Hardware Results Compared

Recall the expected loading in the prototype system. When allocating 4096 items into 32768 hash locations, with no promotion between the hashing levels, the preceding numerical analysis predicts that approximately 40 items will percolate through to the second hashing level. Providing a hash space of 512 locations at this level provides a load factor of better than $\frac{1}{8}$, such that one would expect less than 0.6144 items to percolate through to the final overflow level. Since the numerical model is imperfect for systems of finite size, one would expect the system to exhibit variance about these numerically predicted values.

To characterise this variance in the 2-level architecture, a final software simulation in C++ was written, which periodically reported the loading in both the second level hash space, and the third level overflow memory. The second level results were then processed as before to produce a relative frequency distribution.

To complete the proof-of-concept, a 2-level d-left hardware implementation was written in VHDL and targeted at the prototype platform developed in 2.3.4. The classification logic was fed by a pseudo-random key generator, providing a new key every 16 clock cycles at a system clock period of approximately 6ns (or a system frequency of 170MHz), and left to run for a gating period of 4 hours. Thus with one deletion/insertion event every 16 x 6 = 96ns, the behaviour of the system may be analysed over approximately 1.5 x 10^{11} events.

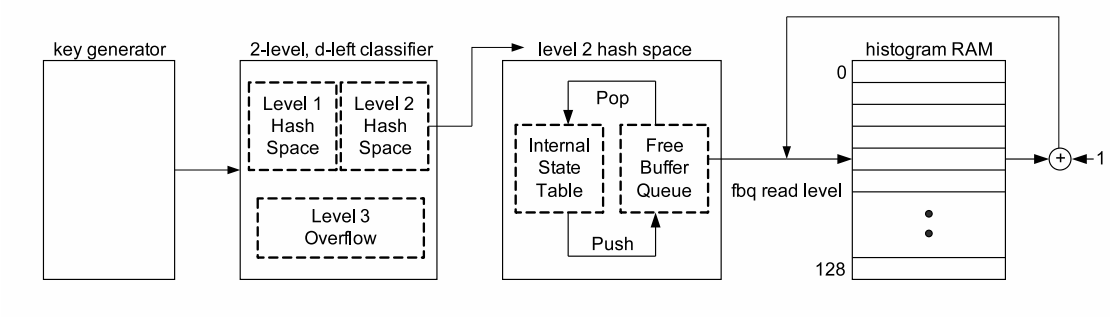

Figure 4.38: *Characterising internal loading in a 2-level d-left implementation*

The allocation of internal state memory in the hardware implementation is controlled by a simple data structure known as a *free buffer queue*. This structure simply holds the available state table addresses and allocates them on demand, on a first-in-first-out basis. When a new item arrives for allocation into the internal state memory, an available location is read or *popped* from the free buffer queue. When an item is deleted, its state table location is freed up and written or *pushed* back onto the free buffer queue as an available location for some future entry. This free buffer queue is implemented in block RAM. One may thus use its associated read level to determine how many items are stored in the level 2 hash space at any point in time, as shown in Figure 4.38.

In the prototype implementation, the free buffer queue read level associated with the level 2 hash space is sampled once every 65536 system clock periods - approximately every 0.4ms. The sampled read level then acts as an address into a histogram RAM. Before running the

system, this RAM is initialised to zero at each of its 128 locations. Once running, when a histogram RAM location is addressed by the classification logic, its contents are incremented by 1. So if the read level is 40, one adds 1 to the contents of histogram location 40. With 32 bits available at each RAM location one can guarantee that it will not overflow during the 4 hour gating period. The values held in this RAM at the end of this period may then be processed to produce a hardware generated relative frequency plot of the loads in the level 2 hash space.

A comparison of software and hardware generated results is shown in Figure 4.39. Good correlation in the level 2 loading is observed between the software simulation and the hardware implementation, and both exhibit a mean load very close to 40, as predicted by the preceding numerical analysis. Finally, the maximum load observed in the third level overflow memory was 1 over 500,000 events in software simulation and 2 over 1.5 x 10^{11} events in the hardware implementation, suggesting that the small overflow space allocated in this system is more than adequate.

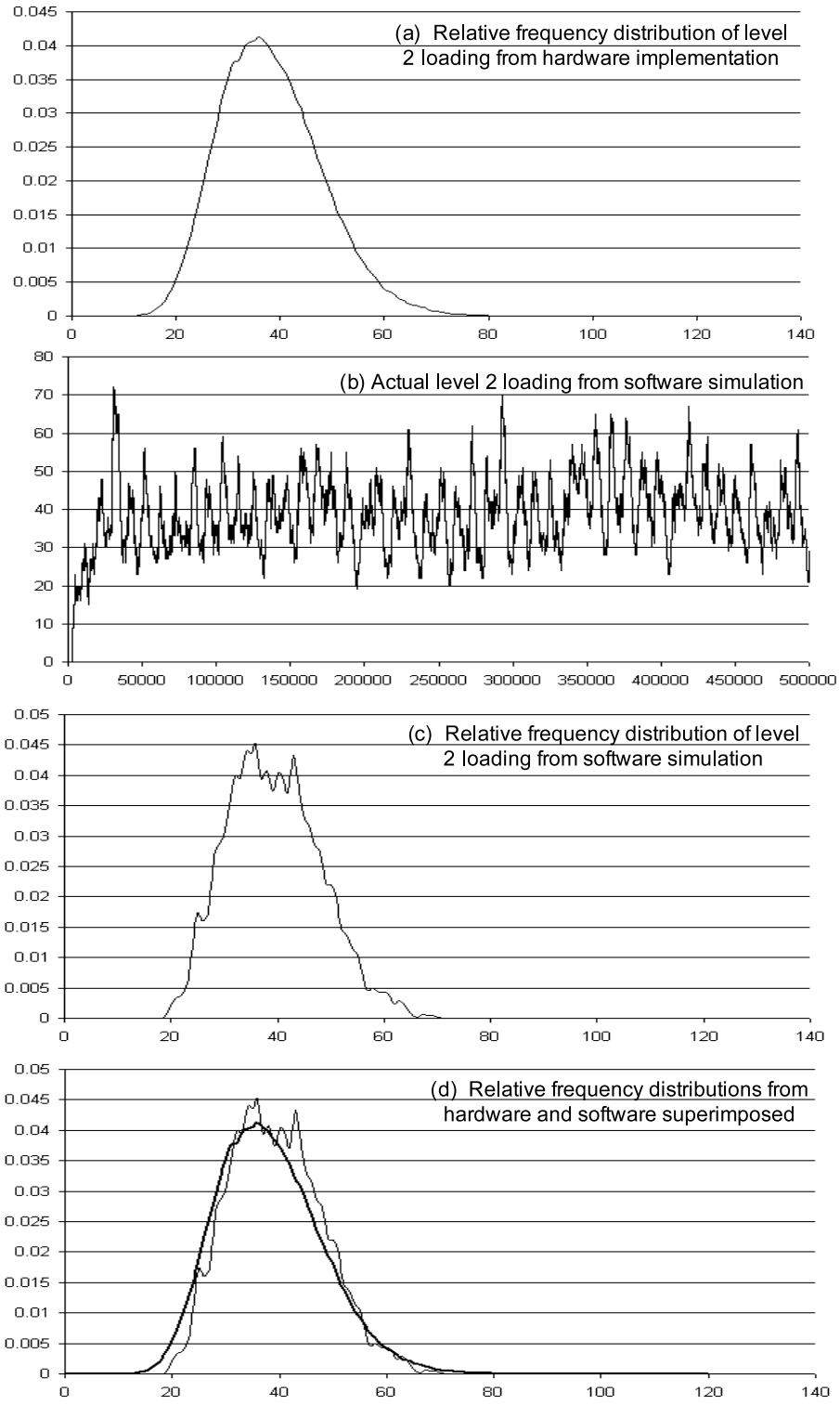

Figure 4.39: *Comparison of software simulation and hardware implementation results for a 2-level, d-left classifier with 4096 input items*

4.7.8 Performance Comparison

Having established the basic viability of the proposed implementation, one now seeks to establish a performance benchmark. The difficulty here lies in deciding what to compare the proposed implementation against. Let us reiterate some of the points discussed in 4.5. Legacy solutions for exact matching are typically limited in terms of lookup performance or implementation efficiency. Techniques based on simple single hashing [95] where all the system entries are stored in external memory, scale efficiently and perform well on average, but very poorly in the worst case - making any claim about high line-rate capability difficult to support. Conversely, techniques based on trees [93] offer guaranteed worst-case lookup times but bind those lookup times to the length of the input item. Further, the exact matching applications of interest here may not be simplified by any heuristic, since there is no *a priori* information about the input items. Tree structures are difficult to optimise as a result.

Based on these foundation techniques, one may thus trade off implementation efficiency and deterministic lookup performance. Whilst tree structure optimisations applicable to exact matching have been suggested [94], attempts to optimise hashing schemes have received more detailed attention in the literature and appear to produce better results. Such optimisation focuses on making hash-based lookup more predictable by reducing the number of collisions which occur by using, for example, cryptographic strength hash functions [100], semi-perfect hash functions [91] or techniques based on multiple hashing, introduced in [101].

Of the latter techniques, the use of Counting Bloom Filters implemented in embedded memory as proposed by Song et al. [106] appears to offer the best results, claiming classification resolution in a single external memory access time. The authors base their results on simulations of a system with $n = 10,000$ input items and $m = 128,000$ external hash locations. To guarantee resolution of a lookup in a single external memory access, one chooses an optimal configuration for their *Shared-Node-Fast-Hash-Table* (SFHT) structure according to the Equation 4.21.

$$k = \frac{m \ln 2}{n} \tag{4.21}$$

In this case, k is the optimal number of hash functions used to address the Counting Bloom Filter. Thus for 10,000 input items and 128,000 external hash table locations, 10 hash functions

are required. The authors show that 3-bit counters in the embedded filter are adequate in this case. The basic system classification logic therefore requires $3 \times 128,000 = 384$ Kbits of embedded SRAM.

Now consider 10,000 input items into the proposed 2-level, 2-left architecture. Allowing 80,000 external hash table locations gives a load factor of $\frac{1}{8}$ in the first level hash space. From the preceding analyses, without any promotion between the hashing levels, one would expect approximately $0.0012 \times 80,000 = 96$ items to percolate through into the embedded second level hash space. Facilitating a load factor of $\frac{1}{8}$ at this level thus requires $8 \times 96 = 768$ embedded hash locations of 10 bits each - a total of 7.68 Kbits of embedded SRAM.

From this second level hash space, one would expect $0.0012 \times 768 = 0.9216$ items to fall into the third level overflow memory. Provision for 8 such overflow locations at 41 bits each requires 328 bits of embedded logic, and would seem reasonable to guarantee overflow sufficiency and lookup resolution in a single external memory access. In total, the basic system classification logic therefore requires just over 8 Kbits of embedded SRAM. So with the caveat that the first level hashing space must be divided in two (and thus requires 2 discrete memories in parallel of $\frac{m}{2}$ locations each to resolve the lookup in a single memory access time), the 2-level, 2-left FPGA implementation proposed here requires 62.5% of the external memory resource, and just 2% of the embedded internal SRAM required by the equivalent Counting Bloom Filter architecture.

Chapter 5
Summary

This research has focused on the development of FPGA-based architectures for Next Generation Communications Networks, specifically addressing two key technologies in high bandwidth networking - Forward Error Correction and Packet Classification. As a preliminary to the concluding discussions, a brief summary of the points already presented now follows.

5.1 Chapter 1

Chapter 1 introduced the concept of System Level Integration as an amalgam of disciplines evolved to manage the complexities of multi-million gate System-on-Chip devices, and to close the associated *productivity gap* - the discrepancy between transistor density and the number of transistors which may be incorporated into a design in a staff month. ASICs, ASSPs and FPGAs were discussed as the principal variants of System-on-Chip device. The advantages of deploying FPGAs in low volume, high complexity applications were discussed; particularly the mitigation of risk as networking standards emerge.

Maintaining competitive advantage by delivering FPGA-based solutions which are smaller and faster than those of their competitors was emphasised as strategically important to the sponsors of this research, Aliathon Ltd. This emphasis was encapsulated in a thesis statement, which asserted that theoretical research and detailed operational study of the algorithms which underpin emerging networking standards, combined with an architectural-level focus on FPGA design, would yield solutions which bettered the state-of-the-art. Forward Error Correction and Packet Classification were introduced as the principal areas for investigation.

5.2 Chapter 2

Chapter 2 presented some technical and commercial background. The term “next generation” was qualified as embodying both a transition to higher bandwidth systems and a migration to

packet-based networking. Specifically, the evolution of the Optical Transport Network (OTN) and the ubiquity of Internet Protocol were discussed as the key commercial motivations for the work presented in this dissertation.

FPGA architectures were discussed, using the Xilinx Virtex II Pro device as an example, and the differences between FPGAs and ASICs/ASSPs were outlined in the context of their respective design flows. The system design trade-offs between logic utilisation, clock speed and processing time were highlighted. *Abstraction* and *technology independence* were outlined as best-practice guidelines followed at Aliathon Ltd.

Finally a description of an FPGA-based configurable network interface card designed as a precursor to the principal research work was presented.

5.3 Chapter 3

Chapter 3 presented the outcomes of the first major research phase of the project, on Forward Error Correction (FEC). A brief introductory discussion outlined reasons why one may wish to encode data for communications. Block Codes were identified as the focus of this research and Finite Field Theory was introduced as the mathematical framework on which the analysis of such codes is based. The Reed-Solomon codes were defined, with RS(255,239) of particular interest as the code deployed in the Optical Transport Network (OTN). The particular challenge of implementing FPGA-based Forward Error Correction for 43Gbps OTU-3 systems was discussed.

5.3.1 Reed-Solomon Encoding

Reed-Solomon encoding was introduced, and the mapping of the required mathematical operations to hardware structures illustrated. The limitations of a single-symbol encoder in the context of OTU-3 systems, which require two symbols to be processed in every system clock cycle, were identified. When using a single-symbol engine, in every clock cycle two input symbols arrive and only one may be processed. Thus to keep up with the incoming data, parallel encoders, data buffers and pipelined logic must be introduced.

A novel two-symbol encoder based on reformulated arithmetic was presented to address these issues. The new encoder is based on a generalised expression relating an arbitrary parity symbol

to two input message symbols, and can thus process two symbols in every system clock cycle - completely removing the need for any interim data buffering.

Since the number of symbols per message word in any Reed-Solomon code is odd, in processing such messages two symbols at a time it is necessary to deal with an odd cycle in every message word, when one of the input symbols to the encoder is non-valid. A wrapper was thus proposed, designed to fit around the encoder core, transparently formatting the data in such a way that the encoder always processes an even number of symbols, but the data remains OTU-3 compliant at the equipment interface.

5.3.2 Reed-Solomon Decoding

The Syndrome Calculator, Chien Search, Forney Calculation and Key Equation Solver block were then introduced as the building blocks of a Reed-Solomon decoder. The Key Equation Solver (KES) block, which takes the syndrome as input and produces the error locator and error evaluator polynomials as outputs, was identified as the most difficult to implement in hardware, since solution of the key equation is mathematically intractable.

Two well documented approaches to the solution of the key equation were discussed - the Berlekamp-Massey algorithm, and the Extended Euclidean algorithm. A contemporary "hybrid" algorithm [12] drawing on the mathematical bases of both approaches was then introduced and identified as an interesting candidate for further study.

Operation of the algorithm was verified in longhand, and tested against known good behavioural models. A previously published VLSI architecture based on the hybrid algorithm was then discussed. It was noted that an FPGA architecture based on a direct implementation of this VLSI model would not compare favourably with Aliathon's existing solution (based on a variant of the Berlekamp-Massey algorithm).

It was thus decided to investigate the operation of the hybrid algorithm further, with a view to identifying possible optimisations. A synthesizable VHDL implementation of a Key Equation Solver based on the hybrid algorithm was developed, and embedded in a test system comprising known behavioural models of the Syndrome Calculator, Chien Search and Forney Calculation blocks. The behaviour of individual coefficients in the evaluator and locator polynomials was then observed on a per-iteration basis until the algorithm terminated, over the full range of correctable error patterns.

From this investigation, two optimisations were proposed. It was noted that the algorithm's working polynomials could be concatenated and normalised to reduce the storage requirements from $8t + 4$ symbol locations to $4t + 6$, where t is the error correcting capability of the code. This optimisation also reduces the number of finite field multipliers required from $8t + 4$ to $4t + 6$ and reduces the number of finite field adders required from $4t + 2$ to $2t + 3$. Further it was noted that modifying the initial conditions allowed the algorithm to complete in $2t - 1$ clock cycles. This improvement, though small in absolute terms, was found to be integral to improved performance at 43Gbps.

A prototype FPGA architecture based on the optimised hybrid algorithm was then developed. The prototype Key Equation Solver can process 1 codeword in 15 clock cycles, or 167 codewords in 240 clock cycles. Therefore with 2 of the new engines, comprising approximately 2000 slices each, the Key Equation may be solved with a total utilisation of 4000 slices. In contrast, a decoder based on the inversionless Berlekamp-Massey algorithm - previously developed at Aliathon Ltd. - requires 32 engines comprising approximately 350 slices each to give a total utilization of 11200 slices. The new implementation thus requires just 36% of the original utilisation logic for operation at OTU-3 line rate.

5.4 Chapter 4

Chapter 4 presented the outcomes of the second major research phase of the project, on Packet Classification. The topic was introduced as a key enabling function for an increasing number of networking applications, including Internet Protocol (IP) routing and switching, Quality of Service (QoS) provision and network security. Three principal types of classification were described - String Matching, Longest Prefix Matching and Exact Matching - the latter being of particular commercial interest to Aliathon Ltd., facilitating the immediate enhancement of legacy products for ATM, and simultaneously creating a platform on which to build a suite of packet-processing IP cores in the future.

The space, time, power and update complexity of classification operations were discussed and a review of previously published classification solutions presented. Existing exact matching techniques were presented, including schemes based on decision-trees, neural networks, Bloom filters and hashing. It was noted that state-of-the-art techniques based on trees offer completely deterministic classification or lookup times, but scale poorly to systems with long headers.

Conversely, contemporary techniques based on hashing scale better but are non-deterministic, such that worst case lookup times can be poor.

Solutions based on multiple hashing were identified as more promising candidates for FPGA implementation. Of these, a structure known as a Counting Bloom Filter and a load balancing algorithm known as *d-left* appeared to perform particularly well. Commercial considerations (in particular the existence of patents covering FPGA-based Bloom Filter implementations), and some interesting gaps in the existing analysis of *d-left* suggested the latter as a good starting point for further investigation.

A numerical analysis of the *d-left* algorithm was then presented. This analysis initially followed the existing published work, based on differential equations, of Mitzenmacher and Broder [16]. Their results for static systems were replicated and extended to include systems with more favourable load factors, where *d-left* was shown to perform significantly better than single hashing. These initial results prompted the following question - given advances in high-bandwidth FPGA embedded memory technology and an appropriately chosen d-left topology to reduce the number of collisions, might it be possible to resolve all these collisions on-chip, and thus create a lookup mechanism offering the advantages of both determinacy and storage efficiency? Additional numerical analysis was deemed necessary to answer this question.

In [16] the authors considered lookup systems where all the items (including those which collide) are nominally stored in external hash memory. The contents of external memory are read into a line in *internal* local cache, where comparison with the queried data occurs. Failure in such a system occurs when collisions cause the data fetched from external memory to exceed the capacity of the local cache line. The *maximum* load in any hash location in the system is thus the critical metric.

It was noted that in proposing to resolve hash collisions in *internal* FPGA memory, the maximum load in any hash bin is not the critical metric - rather the *total* number of items which collide in the external hash space is crucial. The desired system attribute in this regard was defined as *overflow sufficiency* [118]. A new numerical analysis was thus proposed, intended to establish whether *overflow sufficiency* in a dynamic context was actually possible. Initial analyses, supported by a series of software simulations developed in C++, suggested that the amount of FPGA embedded memory required to support a *d-left* classifier was bounded,

reaching a predictable steady state value during dynamic operation at the system capacity.

Attempts to prove the unconditional stability of this system analytically were not successful, since the differential equation resulting from the analysis was non-linear and did not readily yield a solution. It was thus decided to proceed empirically, testing the system under more general modes of dynamic operation, and using real IPv4 data traces as inputs. Repeatable steady state behaviour continued to be observed. On this basis it was decided to proceed with a prototype FPGA implementation.

A discussion of the initial prototype architecture was then presented, and the implications of introducing physical separation between the primary hash space and the overflow memory discussed. The concepts of promotion and memory efficiency were introduced and a revised numerical model proposed to account for them. To maximise the number of classification decisions which could be made in a given test interval, and to remove the need to implement discrete memory controllers, the prototype classifier was implemented entirely in embedded FPGA memory. The basic circuit operation was then described, and the complexities and limitations associated with the architecture discussed.

A novel 2-level, 2-left architecture was proposed as an improved classifier prototype, resolving the difficulties associated with complex comparator logic and caching identified in the earlier system. The prototype was augmented with basic diagnostics and exhibited excellent correlation both with numerically predicted results and software simulations of the underlying random processes. The 2-level, 2-left FPGA implementation proposed requires 62.5% of the external memory resource, and just 2% of the embedded internal SRAM required by a Counting Bloom Filter architecture of equivalent capacity.

Chapter 6
Conclusions and Further Work

The work presented here has considered System Level Integration through the application of programmable logic to some of the design challenges posed by next generation communications networks. Such networks reflect an industry demand for higher bandwidth connectivity compounded by the need to support increasingly diverse payload data. The risks and costs associated with development of fixed functionality devices such as ASICs and ASSPs in this dynamic environment, mean that FPGAs increasingly represent a competitive alternative.

With these points in mind, this dissertation has sought to combine analysis of the algorithms and techniques associated with two important networking paradigms - Forward Error Correction and Packet Classification - with an informed approach to the development of FPGA-based architectures. The thesis statement of Chapter 1 asserted that such an approach could yield solutions which better the state of the art. The Reed-Solomon architectures of Chapter 3 and the classifier prototype of Chapter 4 are offered as justification of this assertion, and in support of the conclusion that the work presented has been broadly successful.

To qualify that conclusion, and by way of final summary, some areas for improvement and suggestions for future work are presented.

6.1 On Forward Error Correction

The logic proposed in 3.5.12 for the optimised Key Equation Solver runs at approximately 170MHz. Whilst this is nominally fast enough to support 43Gbps line rate operation, it allows no margin for the degradation due to system jitter associated with multiple switching gates in near-full FPGAs. Two aspects of the design appear to be critical here. Firstly, there is a large routing delay associated with the δ and γ symbols which must be routed to multiple coefficients to implement the distributed multiplication required by the algorithm. This could be improved by experimenting with register replication to reduce the large fanout on the associated nets.

Secondly, the selection logic combined with the arithmetic blocks form a critical combinatorial path in the design. In particular, the assignment of the b polynomial is determined by an input multiplexer (for initialisation), a combinatorial path through the Galois Field multipliers and adder, and an output demultiplexer which determines whether shifted, non-shifted or zero symbols are passed through.

Experiments with a simplified single coefficient model suggest that the efficiency of the selection logic can be improved by normalising the polynomials in a single stage, as shown in Figure 6.1. This restructuring takes advantage of the fact that the FPGA can implement a 4 to 1 multiplexer in a single CLB, by using an internal construct called an F5 multiplexer. The Key Equation Solver requires further engineering effort to integrate these changes.

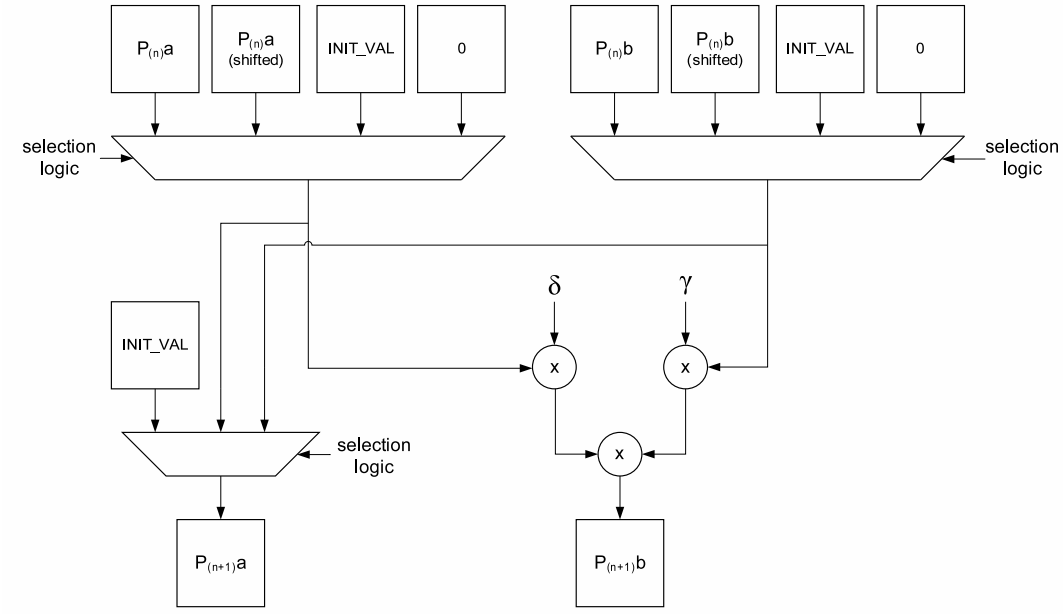

Figure 6.1: *KES architecture for a single coefficient with improved timing*

The Syndrome Calculator, Chien Search and Forney Calculation have not been studied in detail. Further work is required to understand the implications for these blocks in OTU-3 systems, where two symbols must be processed in every clock cycle. The Syndrome Calculator in particular, being similar in basic structure to an encoder, may yield to a mathematical reformulation similar to that applied in 3.5.3. Further, it would be interesting to extend such reformulation to consider systems processing 4 symbols per clock cycle - perhaps as a candidate architecture for an OTU-4 system.

OTU-3 Encoder Engine	Virtex-II Pro Family e.g. XC2VP100			Virtex-IV Family e.g. XC4VLX160		
	Used by architecture	Available in device	Percentage utilisation	Used by architecture	Available in device	Percentage utilisation
Slices	5600	44,096	12.7%	5600	67,584	8.3%
OTU-3 Key Equation Solver	Virtex-II Pro Family e.g. XC2VP7-6FG-456			Virtex-IV Family e.g. XC4VLX15FF668-10		
	Used by architecture	Available in device	Percentage utilisation	Used by architecture	Available in device	Percentage utilisation
Slices	4000	44,096	9.1%	4000	67,584	5.9%

Table 6.1: *Resource utilisation for OTU-3 FEC architectures*

Significant further development work is required to integrate the architectures developed in this dissertation into an OTU-3 chipset. Specifically, for a complete OTU-3 solution, interleaver/de-interleaver, mapper/de-mapper and framer/de-framer cores are all required. Table 6.1 gives some indicative percentage utilisation figures for the architectures developed in this dissertation, which suggest that high-end Virtex II Pro or Virtex IV devices would be good candidate platforms for such a chipset.

Finally, industry adoption of Forward Error Correction in general remains an interesting topic, worthy of ongoing study. The authors of [125] note that the codes considered in this dissertation are optimal for systems spanning less than transoceanic systems, but that for transoceanic applications concatenated codes or product codes offer a better choice. Intriguingly, ITU-T Recommendation G.975 [126] offers multiple suggestions for such strong codes or "super FEC" schemes but does not actually standardise on one - reflecting the uncertainty which remains in this field. Discussion with a number of Aliathon's customers towards the end of the research period documented here, suggests that equipment manufacturers may be adopting the management functions specified by ITU-T G.709, but seeking to implement their own proprietary FEC schemes, optimised for their particular network topologies.

6.2 On Packet Classification

The architecture developed in Chapter 4 represents a promising proof-of-concept around which an FPGA-based classification product could be based. However, further investigation and development work are required to build a genuine product on the ideas presented here. Attempts to derive a complete analytical model of the observed steady-state behaviour of the system were

not successful. Therefore, it was not possible to *prove* that the classifier is unconditionally stable during dynamic operation. The analysis presented in 4.6.10 was not exhaustive, so the existence of such a proof remains an open question.

The system also needs to be more rigorously tested, ideally in a real network environment in parallel with third-party equipment for independent verification of the classification behaviour. The GIGEMON - an open platform for passive monitoring of optical Gigabit Ethernet networks, based on an Endace DAG4.3GE dual channel network monitoring card [127], and the Agilent Technologies N2X Multi-services Tester [128] would provide an excellent test environment - facilitating complete control over traffic generation and comparison logs for traffic capture.

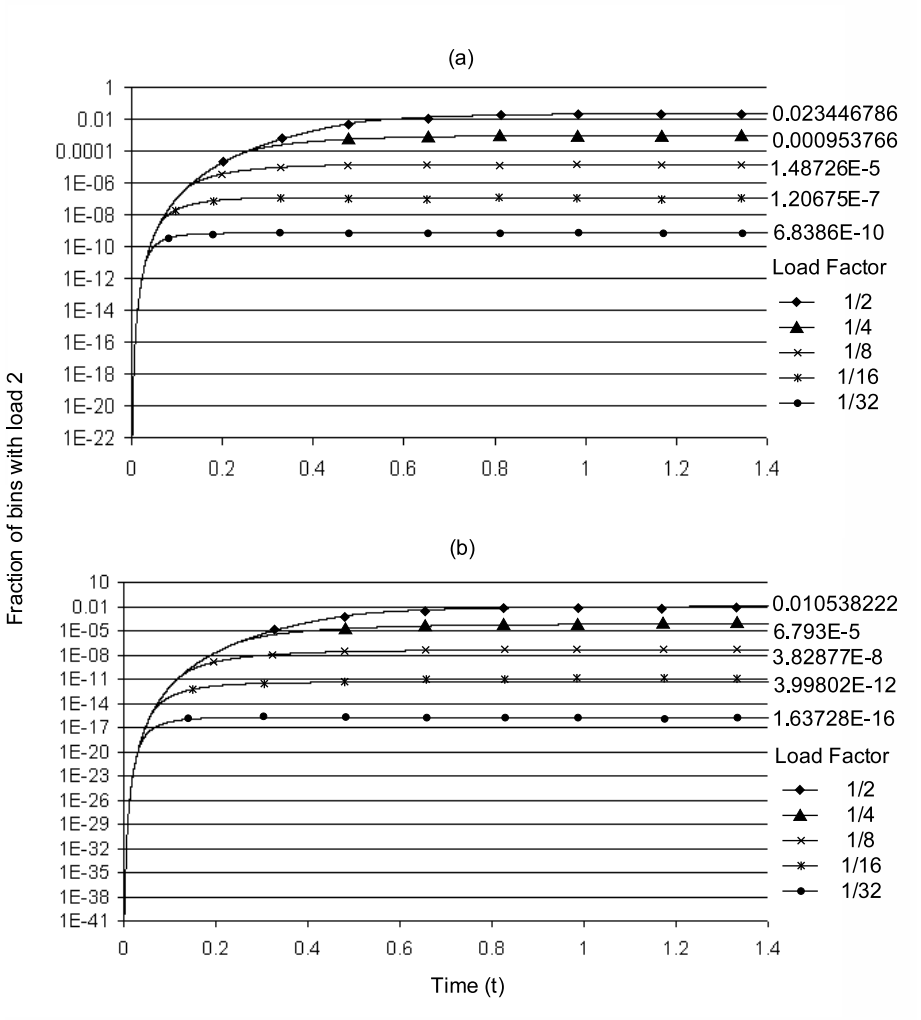

Figure 6.2: *Fraction of bins with load 2 for varying load factor, with continuous deletion and reinsertion at the system capacity M, for a 3-left (a) and 4-left (b) allocation*

The likely performance of real, discrete memory based classification systems based on the proposed architecture needs more detailed analysis. Whilst the system nominally classifies input items in a single external memory access, the actual line-rate capability will depend on the memory technology used. Very large classification systems will most likely be implemented in discrete dynamic RAM; achieving efficient access to such devices is non-trivial and itself worthy of detailed investigation.

The possibility of using higher order *d-left* allocations is suggested. Whilst the benefits of higher order systems follow a law of diminishing returns, a numerical modelling of *3-left* and *4-left* allocations, shown in Figure 6.2, suggests they may also provide a viable basis for alternative FPGA-based classifier topologies. Finally, since the completion of the research documented in this dissertation, additional publications of interest have emerged. The authors of [129] discuss the benefits of allowing items to be moved during insertion into multiple-choice hashing structures in hardware, and in [130] the integration of *d-left* hashing and Bloom Filters is proposed.

Appendix A
An Inversionless Berlekamp-Massey Algorithm

Guide to notation:

- λ_i : Locator polynomial coefficients
- ω_i : Evaluator polynomial coefficients
- δ : Discrepancy, b : Working polynomial, k : Control variable
- γ : Working GF element

Inversionless Berlekamp-Massey (iBM) Algorithm

Initialization:

$\lambda_o(0) = b_o(0) = 1$, $\lambda_i(0) = b_i(0) = 0$ for $i = 1, 2, \ldots, t$. $k(0) = 0$. $\gamma(0) = 1$.

Input: $s_i,\ i = 0, 1, \ldots, 2t-1$.

for $r = 0$ **step** 1 **until** $2t-1$ **loop**

begin
Step iBM.1 $\delta(r) = s_r \cdot \lambda_0(r) + s_{r-1} \cdot \lambda_1(r) + \cdots + s_{r-t} \cdot \lambda_t(r)$

Step iBM.2 $\lambda_i(r+1) = \gamma(r) \cdot \lambda_i(r) - \delta(r) b_{i-1}(r), (i = 0, 1, \ldots, t)$

Step iBM.3
if $\delta \neq 0$ **and** $k(r) \geq 0$ **then**

begin

$b_i(r+1) = \lambda_i(r), (i = 0, 1, \ldots, t)$
$\gamma(r+1) = \delta(r)$
$k(r+1) = -k(r) - 1$

end

else

begin

$b_i(r+1) = b_{i-1}(r), (i = 0, 1, \ldots, t)$
$\gamma(r+1) = \gamma(r)$
$k(r+1) = k(r) + 1$

end
end loop

for $i = 0$ **step** 1 **until** $t - 1$ **loop**

begin
Step iBM.4 $\omega_i(2t) = s_i \cdot \lambda_0(2t) + s_{i-1} \cdot \lambda_1(2t) + \cdots + s_0 \cdot \lambda_i(2t)$
end loop

Output: $\lambda_i(2t), i = 0, 1, \ldots, t.$ $\omega_i(2t), i = 0, 1, \ldots, t-1.$

Appendix B
An Extended Euclidean Algorithm

The algorithm is performed on a table with four columns - quotient $q(x)$, remainder $r(x)$, $u(x)$ and $v(x)$.

If greatest common divisor of two polynomials gcd$(a(x), b(x))$ is given by $r(x)$, the Extended Euclidean algorithm returns polynomials $u(x)$ and $v(x)$ such that $r(x) = u(x)a(x)+v(x)b(x)$.

Definition:

Step 1:
Initialisation. In rows -1 and 0, leave the quotient column empty. The entries in the remainder, $u(x)$, and $v(x)$ columns are $a(x)$, 1 and 0 in row -1 and $b(x)$, 0 and 1 in row 0. Set iteration count $k = 0$.

Step 2:
Calculation of $q(x)$. Divide $r(x)_{k1}$ by $r(x)_k$ producing the quotient $q(x)$ and the remainder $r(x)$.

Step 3:
Calculation of $r(x)$, $u(x)$ and $v(x)$. The formulae determining $r(x)_{k+1}$, $u(x)_{k+1}$ and $v(x)_{k+1}$ are:
$r(x)_{k+1} = r(x)_{k1}q(x)_{k+1}r(x)_k$
$u(x)_{k+1} = u(x)_{k1}q(x)_{k+1}u(x)_k$
$v(x)_{k+1} = v(x)_{k1}q(x)_{k+1}v(x)_k$

Go to Step 2. Terminate when the degree of $r(x)$ is less than t, where t is the error correcting capability of the code. For details of failure conditions when this capability is exceeded see [36].

The algorithm returns the error locator and error evaluator polynomials when the inputs are $a(x) = x^{2t}$ and $b(x) = s(x)$.

Appendix C
An Outline Business Plan

A business plan submitted to the University of Glasgow in fulfilment of the business and management requirements of the Engineering Doctorate.

Declan Hegarty
Institute for System Level Integration

Aliathon Ltd.
Evans Business Centre
Pitreavie Court
Dunfermline
Fife
KY11 8UU

Registered Company SC216137
Tel + 44 (0)1383 737736
Fax + 44 (0)1383 749501
www.aliathon.com

C.1 Executive Summary

- Aliathon is a dynamic Scottish start-up company, founded in 2001, delivering world-class IP core products to global telecoms and datacoms 'markets.
- These markets will be worth some $20 billion by 2008.
- Aliathon's proven business model is beginning to establish market share; poised to deliver 40% revenue growth in the next financial year by building on an established international customer base.

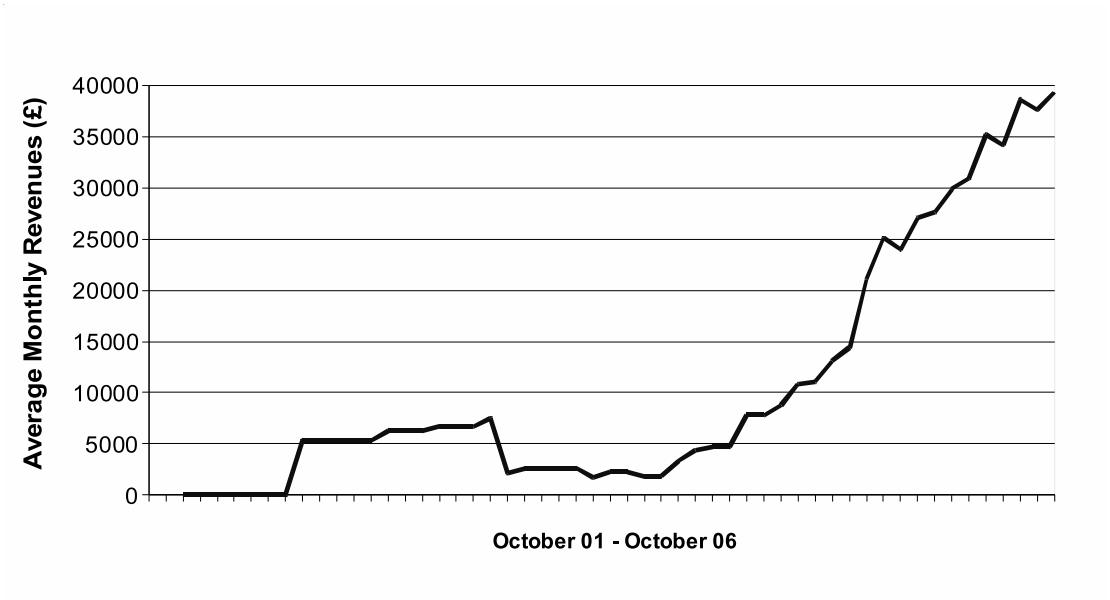

Figure C.1: *Aliathon revenue growth 2001-2006*

- Aliathon operates in the high value-added microelectronics sector, with the potential to generate additional high quality jobs for Scotland and bring a significant revenue stream into the Scottish economy.
- Aliathon is thus an excellent fit with the targets, key markets and technologies identified in Scottish Enterprise's unified strategy for microelectronics and optoelectronics for 2005-2009 [131].
- Aliathon now seeks engagement with Scottish Enterprise to evaluate ways to consolidate and accelerate growth of the business. Principal areas for discussion and development

include the development of an appropriately scaled marketing mix, consolidation of existing strategic partnerships with Xilinx, Inc. and Altera Corp., and support for ongoing research and development. Other areas of interest include the protection of strategic intellectual property, management training and assistance with executive appointments.

C.2 Company Overview

C.2.1 Introduction

Aliathon designs, develops, verifies and sells Intellectual Property (IP) solutions for the communications industry, targeting an advanced family of chip devices known as Field Programmable Gate Arrays (FPGAs). In this context, Intellectual Property refers to the design blocks or cores which run inside these chips, comprising a global market estimated to be worth $1.27B in 2004 [132]. Such IP cores may be combined to provide complete silicon chip solutions for a wide range of communications and networking applications.

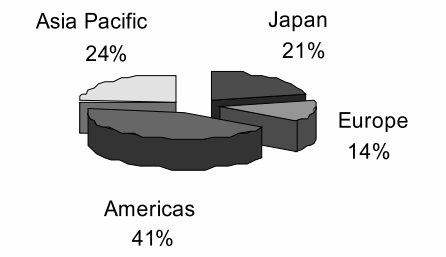

Figure C.2: *Total global IP revenue, 2004 - $1.27 Billion*

In general, electronic equipment typically comprises one or more printed circuit boards (PCBs) which host and interconnect a number of discrete integrated circuits (IC) designed to perform a specific function. Gordon Moores famous prediction from 1965 [1] correctly stated that the transistor density (and thus processing power) of such integrated circuits would double approximately every eighteen months.

The communications chip market today is predominantly addressed by high performance FPGAs, Application Specific Integrated Circuit (ASIC) and Application Specific Standard Products (ASSP) solutions, capable of integrating many functions (including high speed wireline and wireless capabilities) on a single chip, thus replacing multiple legacy devices and reducing system cost.

Gartner Dataquest estimate this market to be worth some $30 billion, growing to $45 billion over the next five years.

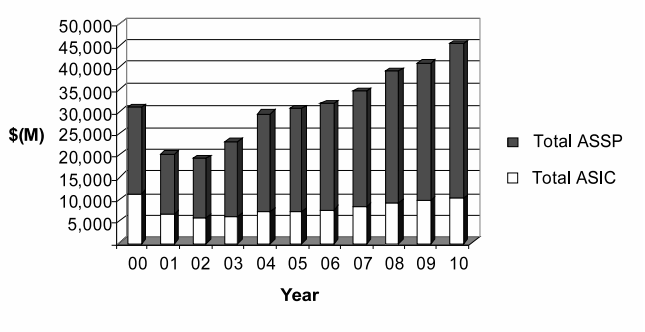

Figure C.3: *Market for communications chip solutions 2000-2010*

Aliathon designs, combined with leading edge FPGA technologies will continue to displace the incumbent solution providers and win market share in this timeframe for the following reasons:

- ASIC/ASSP development is very expensive, at up to $30 million for modern devices [9]. Aliathon can reduce the overall system design cost to end users.
- FPGA designs can be implemented faster than ASICs/ASSPs. Aliathon can thus reduce time to market for end users.
- FPGAs are completely reprogrammable. Evolving standards and customer requirements can therefore be easily tracked. Aliathon can thus reduce the risks of new product development for end uses.

C.2.2 Operational Summary

- Aliathon is a limited company SC216137, registered at 10 Craighouse Place, Saline, Fife, KY12 8TQ, Scotland.

Location of Principal Operations

- Aliathon will continue its current operations at Evans Business Centre, Pitreavie Court, Dunfermline, Fife, KY11 8UU, Scotland.

Main Operational Activities

- Development Engineering: Including both work to expand our product portfolio, and design activities to extend or enhance existing products.
- Customer Support: Including technical post sales support (included in the price of the core), maintenance and bug fixing.
- Sales and Technical Marketing: Including detailed technical and commercial discussion of new opportunities with FPGA partners and potential customers.

Current Staffing Complement

- Mr. Steve McDonald, Director
  R&D Architecture and Implementation, Sales & Marketing, Finance, HR
- Mr. Jed Martens, Director
  R&D Architecture and Implementation, Sales & Marketing, Finance, HR
- Mr. Ulises Hernandez, Senior Design Engineer
  R&D Architecture and Implementation
- Mr. Declan Hegarty, EngD Research Engineer *Research and Development*
- Mr. Peter Sinka, EngD Research Engineer *Research and Development*
- Mr. Gavin Fleming, Management Consultant
- Mr. Kevin Dineley, Software Consultant

Infrastructure Requirements

- Aliathon's premises currently comprise two managed offices at the Evans Business Centre, Dunfermline. Aliathon's current offices offer adequate accommodation for two additional employees and additional engineering equipment. Additional office space is readily available within the building.
- Aliathon's development work is carried out using high end simulation and synthesis tools to support system development. The majority of Aliathon's business correspondence is

transferred electronically, including the distribution of IP cores to customers. Office facilities include telephone, fax and broadband internet connectivity.

C.2.3 Industry Overview

Technological advance and significant growth in the Internet, wireless technology and broadband adoption are fuelling the communications industrys return to growth.

New applications and technologies such as wireless internet access, voice-over-Internet Protocol (VoIP) telephony, virtual private networking and third generation (3G) mobile services are generating huge demand for bandwidth and improved quality of service. As a result, the traditionally disparate disciplines of voice and data communications are converging to deliver networks capable of providing what are becoming known as the triple play services: voice, video and data.

Equipment deployed in the Access, Enterprise, Metro and Core networks (see section C.5.2 for more detail) to provide these services is thus becoming much more complex as operators seek to accommodate high volume and a wide range of traffic types. Equipment manufacturers thus face increasing pressure to source Integrated Circuit solutions from expert designers.

Aliathon is ideally positioned in the industry value chain as presented by Thomson and Strickland [133] to deliver these solutions, with a growing portfolio of sophisticated communications products supporting both legacy and next generation network equipment.

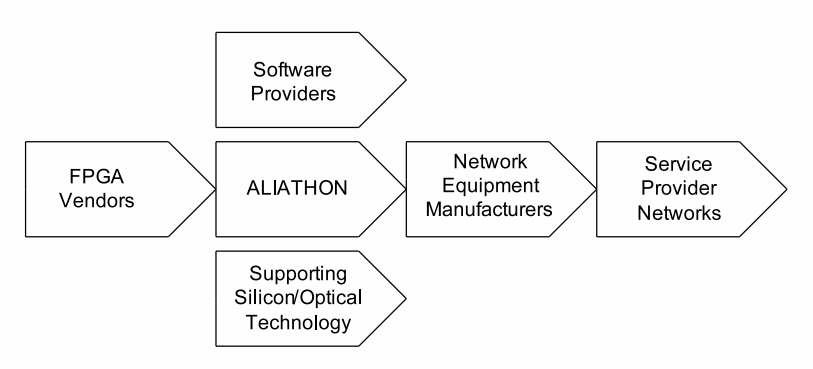

Figure C.4: *Aliathon's position in the communications/networking value chain*

C.3 Products

C.3.1 The FPGA Advantage

Many Integrated Circuits (ICs) for communications applications are extremely complex devices. Traditionally, the market has been dominated by Application Specific Integrated Circuits (ASICs) and Application Specific Standard Products (ASSPs). ASICs are silicon chips designed exclusively for one customer, usually encompassing a range of functionality focused on addressing the requirements of a particular standard or protocol. ASSPs are typically of similar complexity, but marketed as part of the chip vendors portfolio, and usually purchased by more than one end customer.

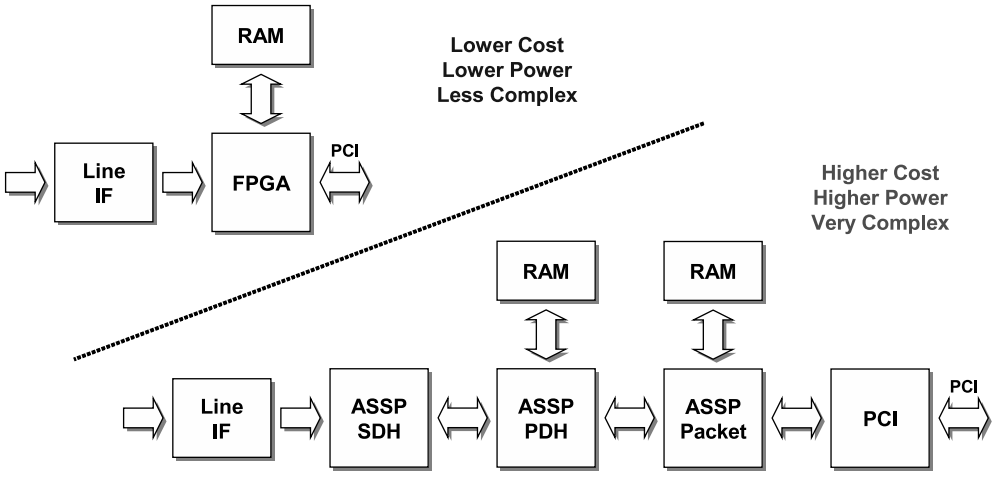

Figure C.5: *FPGA versus ASIC/ASSP in communications systems*

ASICs and ASSPs are known as *fixed functionality* devices. They are hardwired internally and their behaviour cannot be modified once they have been manufactured. Thus, ASIC and ASSP vendors seek to accommodate the widest possible range of functions to attract a wide customer base. Typically this results in a device with a wide range of functions of which only a subset will be used by any given customer, and thus a device which is larger, more power hungry and more expensive than it needs to be.

In contrast, FPGAs are completely reprogrammable devices, allowing easy customisation on a user-by-user basis. Aliathon's customers can therefore enjoy the benefits of reduced cost, power, and time to market afforded by our highly optimised solutions. A wide range of FPGA platform devices are available from the market leaders in the industry, Altera Corp., and Xilinx, Inc. Aliathon has recently secured preferred partner status with both these suppliers, and our IP

is designed to run seamlessly on any of their FPGAs, giving our customers a wide range of IC platforms to choose from.

C.3.2 Product Portfolio

Aliathon's current product portfolio and roadmap is shown below, with telecoms and datacoms IP shown as distinct product segments (some historical background is provided in C.8.1). The transmission speed applicable to each product is shown on the vertical axis, with the product timelines on the horizontal axis. Thus, the bottom left of each figure shows lower speed products available today , the top right showing the faster, emergent technologies still under development. A complete product technical taxonomy is given in C.8.2.

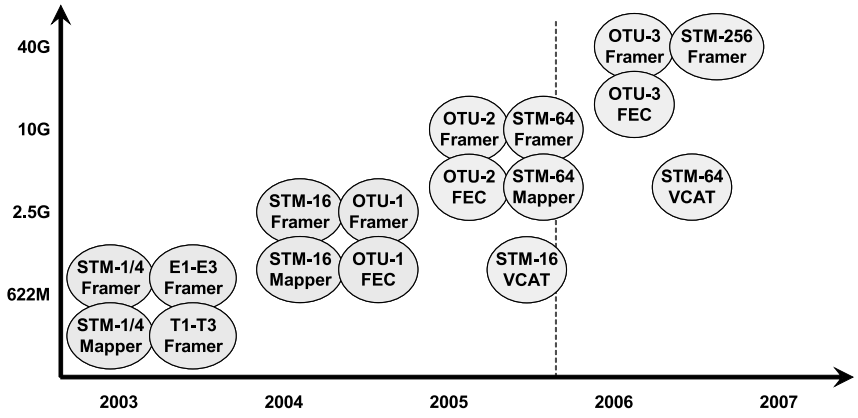

Figure C.6: *Aliathon portfolio and roadmap for telecoms IP*

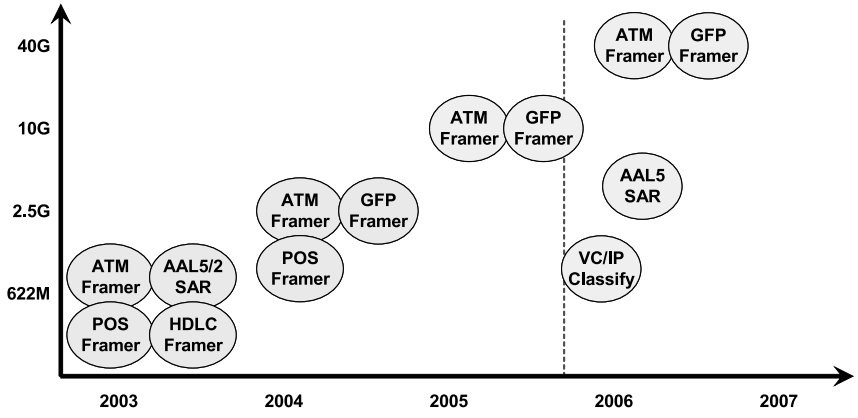

Figure C.7: *Aliathon portfolio and roadmap for datacoms IP*

	Customer								
Core	Agilent	Endace	Alcatel	CarrierCOM	Lucent	Siemens	Sparta	Spirent	NetQuest
T1 Framer	√	√	√						
T1 Deframer	√	√	√				√		
T2 Deframer							√		
T3 Framer	√	√							
T3 Deframer	√	√					√		
STM1/4 Framer	√	√		√				√	√
STM1/4Deframer	√	√		√			√	√	√
Mapper	√	√		√					√
Demapper	√	√		√			√		√
OTU-1 Framer					√	√			
OTU-1 Deframer					√	√			
POS Framer		√							
POS Deframer		√					√		√
ATM Framer		√							
ATM Deframer	√	√					√		
GFP Framer						√			
GFP Deframer						√			
HDLC Deframer	√						√		

Table C.1: *Aliathon customer matrix*

C.3.2.1 Current Customer Matrix

Aliathon boasts a customer base which has grown since the company began trading to include some of the worlds leading communications companies. Product sales to date are summarised in Table C.1

C.3.2.2 Target Customers

Aliathon's target customers are principally Network Equipment Manufacturers (NEMs) or the various complementary enterprises which go along with them, such as network monitoring and test equipment suppliers. These clients supply service providers with the infrastructure and test capabilities required to deploy their networks and services.

The "Tier 1" companies profiled in Aliathon's recent market analysis [134] were Alcatel, Cisco Systems, Huawei Technologies, Siemens, Ericsson, Lucent Technologies, Nortel Networks, Fujitsu, Nokia, Juniper Networks and Marconi. These companies carry significant influence in the communications industry and typically steer the standards bodies upon which the majority of industry activity is based. However, they represent only a small subset of Aliathon's potential customer base.

As a complement to this high level industry analysis, Aliathon has developed an initial targeting matrix of some 130 potential clients, from which the estimated revenue opportunity is over $21 million based on Aliathon cores already developed. Aliathon's ongoing new product development will increase both the breadth of this client base, and its revenue potential. The complete target customer matrix is shown in Addendum III.

C.4 Competitive Environment

Aliathon faces competition for sales to Network Equipment Manufacturers (NEMs) on two fronts; from ASIC and ASSP providers who supply their customers with Integrated Circuit products and whom we seek to displace and from peer providers of Intellectual Property who enable their end customers to either design and manufacturer their own ICs, or produce FPGA solutions.

C.4.1 ASIC/ASSP Competitor Profiles

The ASIC and ASSP providers profiled here are well financed, have significant communications technology, have established sales and distribution channels and enjoy close relationships with the incumbent equipment manufacturers. Importantly, they depend on the markets in which Aliathon operates (or seeks to) for large percentages of their revenue, and are thus given detailed attention in Aliathon's market analysis. A brief summary profile of each competitor is included here.

The wireline market is dominated by Broadcom and Intel who generate much of their revenue from the consumer broadband and physical layer IC segments. These segments are subject to rapid price erosion and commoditization risk, and operate on economies of scale with which Aliathon (and fundamentally the baseline cost of most FPGAs) cannot currently compete.

Aliathon's value proposition is strongest in the high end communications applications where we can most easily differentiate. The incumbent suppliers here are Agere Systems, Infineon Technologies, Vitesse Semiconductor, PMC-Sierra and AMCC. These competitors together accounted for 17% of the wireline ASIC/ASSP market, worth $6.538 billion globally in FY2004[1]. By combining the latest FPGA technologies with our leading edge IP, and leveraging the marketing and sales capabilities of our FPGA partners, Aliathon is extremely well positioned to displace ASICs and ASSPs from next generation network equipment.

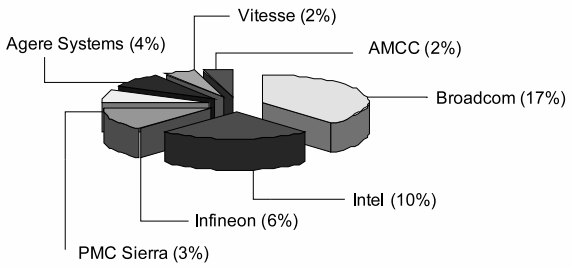

Figure C.8: *Wireline ASIC/ASSP Market, 2004 - $6.5 Billion*

Agere Systems

Agere design, develop, manufacture and sell integrated circuit solutions for the high density storage, mobile wireless, enterprise and telecommunications networks markets. In fiscal 2004, they realigned their business into operating segments tailored to the markets in which they operate. Ageres total revenues were $1,839M in FY2003, rising to $1,912M in FY2004, a 4% increase year-on-year. Of that revenue, $635M came from storage, $469M from mobility, $513M from Enterprise and Networking and $268M from telecommunications.

Infineon Technologies

Infineon is the semiconductor spin-off from its parent company Siemens, and was established as a public company based in Munich in April 1999. Infineons operations are organised into the following business groups: Wireline Communications (COM), Secure Mobile Solutions (SMS), Automotive and Industrial (AI) and Memory Products (MP). Infineons net revenues were 6,152M in FY2003 rising to 7,195M in FY2004, when the company returned to profitability for the first time since FY2000.

[1]Source: Altera Corp., Communications ASSP Market Share Figures.

Applied Micro Circuit Corporation (AMCC)

AMCC design, develop, market and support integrated circuits for the communications and storage equipment markets. AMCCs revenues were $131M in FY2004 rising to $254M as of March 2005, of which 50.8% came from communications, 25.4% from embedded products and 20.3% came from storage. Geographically, 48% of AMCCs revenue came from the U.S., 19.8% came from Europe and Israel and 24% came from Asia Pacific. Sales to Nortel Networks accounted from 11% of total revenues for FY2004.

PMC-Sierra

PMC-Sierra design, develop, market and support high-speed broadband communications and storage semiconductors and MIPS-based processors for service provider, enterprise, storage and wireless networking equipment. PMC-Sierras revenues were $249M in FY2003 rising to $297M in FY2004. They cite an inventory work off period in 2H04 as negative impacting revenues during this period, though much less significantly than during equivalent phases in 1999 and 2000.

Vitesse Semiconductor

Vitesse is a supplier of high performance ICs targeting manufacturers in the communication and storage industries. Vitesse products target the enterprise, metro and core segments of the network. Vitesses revenues were $156M in FY2003 rising to $218M in FY2004. The company has been running at an operational loss since FY2000, and implemented significant restructuring programmes in 2001, 2002 and 2003.

C.4.2 IP Provider Competition

In addition to the principal ASIC and ASSP silicon vendors profiled above, Aliathon faces a competitive threat from various providers of IP, targeting a range of device technologies. These competitors range from large, publicly listed companies like the FPGA vendors themselves, to smaller specialist IP design houses.

Obtaining accurate information on the latter group of companies is difficult. Aliathon must continue to monitor the progress and strategies of all these companies to position itself effectively in the future.

Altera and *Xilinx*

Altera and Xilinx dominate the market for programmable logic devices, and have significant resource available for IP development. Both companies offer IP which competes directly with Aliathon's products. Aliathon has to date differentiated itself by offering superior technical performance, but may suffer pricing pressures in the future from these vendors given their triplicate role of supplier, partner and competitor.

Modelware

Modelware provide a wide range of IP cores for both ASIC and FPGA technologies in the telecommunications and networking markets. Their products currently address ATM, packet interfaces, Ethernet and HDLC and their current bias is towards lower value, higher volume generic interface IP. However, combined expertise in FPGA and communications means that they face lower barriers to entry into Aliathon's high value markets than many other competitors.

CG-CoreEl

CG-CoreEl are a programmable solutions and design services provider headquartered in Bangalore, India. They offer turnkey product development for PCBs, FPGAs and ASICs. The global success of companies based in India, Asia-Pacific and China has traditionally been built around a low cost business model. Such companies are leveraging these successes to increase their innovative capacity. CG-CoreEl thus remain a direct competitive threat, although their current standard IP portfolio does not compete directly with Aliathon's.

Innocor

Innocor is predominantly a provider of telecom and datacom manufacturing test equipment, but offer an expanding portfolio of IP solutions including GFP controllers and ATM solutions.

Flextronics

Flextronics is a multinational design services company headquartered in Singapore, with revenues totaling $15.9B for year ending March 2005. They offer a huge range of system design services including ASIC and FPGA services. Recent acquisition of key assets from Nortel Networks means they are well positioned to development competitive products in telecommunications and networking.

Conexant Systems

Conexant Systems, Inc., design, develop and sell communications semiconductor systems solutions including ICs, software and reference designs. Conexant recently completed the acquisition of Paxonet, and thus own a broad portfolio of IP which competes directly with Aliathon's. Conexants positioning of these products within its wider portfolio is not yet clear.

C.4.3 SWOT Analysis

Aliathon's position within its competitive environment may be summarised by an analysis of the strengths, weaknesses, opportunities and threats [135] which our market and competition present.

C.4.3.1 Strengths

- *Technical expertise*: Aliathon has a proven track record in delivering superior FPGA designs to demanding timescales.
- *Self-funding*: Aliathon has an established revenue stream which is independently funding our ongoing development work.
- *A global customer base*: Including some of the worlds leading equipment manufacturers.
- *An established portfolio of products*.
- *An ability to demonstrate these products working in real hardware*: This is a key differentiator for an IP company, allowing us to demonstrate conformance to standards and interoperability with third party equipment.
- *Low marginal costs for sales of existing products*.
- *Aliathon IP cores are technology independent*: All our products are designed to run on any FPGA technology. This brings significant value to our customers, who can chose an optimal device from a broad catalogue.

C.4.3.2 Weaknesses

- *Limited resources*: Relative to our larger competitors Aliathon has a small design team, and limited access to cash for development work, sales and marketing activities.
- *Relatively low profile within the industry*.
- *Lack of ISO9000 certification, formal processes*: This may limit our credibility in the eyes of larger enterprises. In general, Aliathon's working processes are not formalised or thoroughly documented. IP source code is not always comprehensively commented, particularly where projects are subject to time pressures.
- *Marketing and sales activities and direct engagement with customers are limited*.
- *Aliathon's website needs to be updated*: We need to improve the appearance and functionality of our web presence, and integrate customer support capabilities, such as bug tracking and dedicated customer pages. The quality of online documentation needs to be improved.
- *Aliathon's pricing strategy is ad-hoc*: We need to conduct further analyses to better understand the value of our products to the end user, and formalise our pricing structures to leverage maximum value.

C.4.3.3 Opportunities

- *Displacing ASICs and ASSPs in communications equipment*: With ASIC and ASSP development costs approaching $30M at the 90nm process node, replacing such devices with FPGAs is an increasingly attractive proposition for NEMs.
- *Developing our partnerships with Xilinx and Altera*: This offers huge potential for new design wins, cooperative system design projects and building relationships with key customers.
- *Marketing*: This side of Aliathon's business is currently underdeveloped. Raising our profile and actively selling our products offers a major growth opportunity.
- *Exploiting new technologies*: Aliathon's small size can be an advantage, allowing us to respond quickly to new market opportunities. Emerging technologies currently of interest include ATCA, NG SONET/SDH, GFP, OTN and Ethernet/IP/MPLS.

- *Migration towards converged networks*: Exemplified by BTs progress towards the 21st Century Network. Aliathon holds significant expertise in design for complex communications systems. New technologies will offer ongoing opportunities to leverage these skills in new markets.

C.4.3.4 Threats

- *Partnerships with Xilinx and Altera*: Partner programmes continue to generate significant interest in Aliathon's products. Addressing this interest is time consuming, and diverts focus and effort.
- *Legal threats from competitors and customers*: As an IP business, Aliathon faces the possibility of legal challenge from our competitors in the event that our designs infringe existing patents. Customers may also seek indemnity in this case. Aliathon also needs to protect its own IP.
- *Direct competitive threat from ASIC, ASSP and IP providers*.
- *Technology migration away from legacy SONET/SDH*.
- *Overload*: Closely tied in with our Xilinx and Altera partnerships, and our small design team, is the risk of overloading our limited resources.

C.4.4 Aliathon and the Five Forces Framework

Although some aspects will overlap with those already identified in the preceding SWOT analysis, it is nonetheless helpful to consider Aliathon's competitive position within the framework of another seminal strategic model that of Porters Five Forces of Competition [136]. In applying such a framework to Aliathon's competitive position it is important to understand some of the models limitations.

Grant [137] has criticised the static nature of the model in assuming a stable and externally influenced industry structure. This is certainly a limitation with respect to the electronics industry, where competitive dynamics shape the industry. New entrants with superior technologies can effectively create a market where none previously existed. Grant also questions the validity of direct correlation between environment and profitability. Additionally, we can extend Porters model to consider a sixth force - that of complements [137, 138].

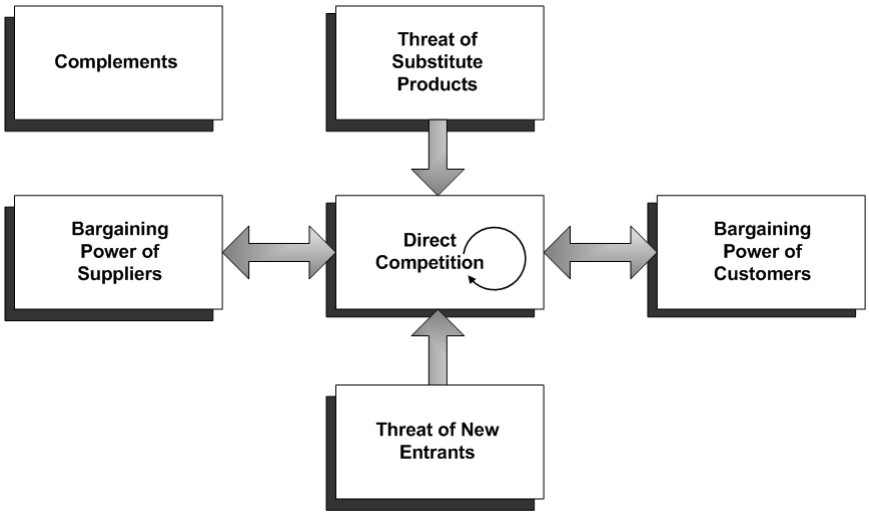

Figure C.9: *Porter's forces of competition*

C.4.4.1 The Threat of New Entrants

The capital barriers to entry into Aliathon's markets are relatively low; in theory anyone with adequate resources to support the design flow for FPGA IP could begin developing such products. Similarly, economies of scale do not present a significant barrier to entry for potential new entrants since high end communications systems are shipped in relatively low volume and command high value.

The most significant barrier to entry is the level of technical expertise required to design and develop competitive silicon solutions. This expertise comprises both a detailed understanding of legacy, current and emergent communications standards, and expert chip design capabilities. Constant pressure to deliver smaller, faster and more efficient solutions mean that only the most capable design teams can compete. However, given the dynamic nature of the industry with frequent divestiture and acquisition of intellectual property and expertise, the competitive threat from new entrants needs to be regularly evaluated, and Aliathon's competitive analysis kept up to date. Another significant barrier to entry for would-be competitors is Aliathon's partnership with both Xilinx and Altera, which secures a valuable distribution and marketing channel.

C.4.4.2 The Bargaining Power of Suppliers

For Aliathon's core business, this competitive force is not really quantifiable in the context originally intended by Porter, since the development of IP cores is a primary input to the design of communications systems, and not dependent on any supply chain. Whilst Xilinx and Altera supply the FPGAs on which Aliathon IP will ultimately run, they are not principal suppliers to Aliathon, since it is our mutual end customers who purchase the devices. Aliathon may be subject to influence by suppliers of Electronic Design Automation (EDA) tools used to develop its products. Overall however, the bargaining power of suppliers is low.

C.4.4.3 The Bargaining Power of Customers

Product differentiation is again of fundamental importance in mitigating the bargaining power of our customers, and leveraging maximum value from our products. Aliathon cores must continue to outperform competitive IP core offerings in terms of silicon area used, power, speed and density. In competition with ASIC/ASSP providers, Aliathon cores must continue to offer a clear cost advantage to the customer.

Aliathon faces pressure both from end-customers directly, and from Xilinx and Altera, who may seek to incentivise strategic customers to purchase their FPGAs by offering artificially low prices for IP solutions.

C.4.4.4 The Threat of Substitution

Replacing ASICs and ASSPs is a core part of Aliathon's business development strategy. Substitution is therefore both a threat and an opportunity. On one hand, Aliathon must seek to leverage the support of our FPGA partners to promote these devices as ideal substitutes for ASICs and ASSPs in communications systems; on the other, Aliathon must continue to produce highly competitive designs to prevent FPGAs being substituted by ASICs or ASSPs in existing designs. The former is the more significant challenge, as established ASIC and ASSP users often need to be carefully educated and convinced of the merits of FPGA technology if they are to switch.

C.4.4.5 Direct Competition

Aliathon's overall competitive position is summarised in C.4.1 and C.4.2. Aliathon must continue to differentiate its products, providing superior quality, performance and cost benefit to the end user.

C.4.4.6 Complements - A Sixth Force

Chip solutions for the communications industry comprise more than just the core hardware design itself. The design needs physical hardware to run on. Thus the FPGA providers, Xilinx and Altera are vital complements to Aliathon's business. Aliathon IP adds significant value to the FPGA providers, since they can present their target customers (and thus Aliathon's) with proposals for complete hardware solutions a much stronger value proposition than merely offering empty platform FPGAs.

Chip solutions require a register interface and driver layer software to manage the contents of these registers. Application layer software and a user interface are also required to make chips usable. These represent key complements to Aliathon's core products the more complete the solution the more valuable it is to the end customer.

An ongoing part of Aliathon's strategy is to pursue synergistic partnerships with providers of complementary products, such as software drivers. The partnership programmes of Xilinx and Altera represent an opportunity for collaboration with these companies and their other partners on a project by project basis. However, the competition between these vendors is fierce, and they will seek to consolidate competitive advantage by extracting maximum value from Aliathon as a partner company when pursuing opportunities. This can place significant strain on our resources.

C.5 The Market Environment

C.5.1 PESTLE Analysis of the Macro-Environment

C.5.1.1 Political Influences on Aliathon

Aliathon's opportunities and ability to compete locally and globally are subject to a number of political influences of varying scope. Fundamentally Aliathon operates within a climate of

relative political stability. However, geo-political instability in other regions may negatively impact levels of capex in the markets to which Aliathon currently sells its products. Additionally and in common with other enterprises, Aliathon's operating margin is directly impacted by taxation levels. Such generalised factors constitute the political fabric within which Aliathon must operate.

Whilst such factors are important, political decisions which filter directly into the communications industry are potentially of greater interest. Many sectors of the communications industry are moving towards deregulation, presenting a significant threat to incumbent operators in the industry, and an opportunity for smaller enterprises to compete. For example, Nippon Telegraph and Telephone (NTT), an incumbent operator in Japan, has been forced to reduce tariffs due to increased competition following deregulation. Markets based on emergent technologies such as Fibre-to-the-Home (FTTH) are set to follow similar deregulated models. Washington has additionally unveiled the Broadband Consumer Choice Act 2005 paving the way for sweeping deregulation of broadband-based applications. This will incentivise challenge to incumbent suppliers and ultimately help to broaden Aliathon's customer base.

Federal mandates and government or commission directives which demand provision of key technologies are also of significant interest to Aliathon. The Federal Communications Commission (FCC) has recently introduced the Enhanced 911 directive (E911) which sets rules for the provision of reliable emergency services by wireless providers. This has stimulated significant activity in mobile location technology development. In the internet domain, a recent U.S. Department of Defense (DoD) mandate has stipulated that all Information Technology (IT) assets procured after 1st October 2003 should be Internet Protocol version 6 (IPv6) capable. IPv6 is likely to provide the backbone for many of the converged triple play services required in next-generation-networks (NGNs), but adoption since standardisation has been slow. Mandates such as that issued by the DoD, which represents a potential $50 billion per annum market opportunity, are thus likely to stimulate technological development directly.

North American resistance to wider adoption of IPv6 is in part borne of the fact that 70% of the available IPv4 address space[2] has been allocated there. Political influence on adoption of IPv6 infrastructure is thus likely to come from the European and Asia-Pacific theatres. The Japanese government has already issued a number of high profile statements outlining its commitment

[2]IPv4 uses 32 bit addresses, most of which have been consumed. IPv6 uses 128 bit addresses.

to IPv6. A formal mandate is likely to follow. British Telecoms 21st Century Network (21CN) plans represent another significant commitment to the technology.

The Asia-Pacific region represents a significant growth opportunity for the communications industry, with extremely large populations in countries such as India and China, served by a comparatively poor established network infrastructure. International political relationships will play a key role in determining how these opportunities are realised. China and India have both been cited by the U.S. Trade Representative as upholding unfair trade barriers to telecommunications imports, with excessive regulatory requirements and unfair protection for native equipment manufacturers. The U.S. strategy to enforce its right to open competition in these regions could ultimately include retaliatory trade sanctions, which may impact Aliathon's ability to build business relationships with enterprises in both North America and Asia.

C.5.1.2 Economic Influences on Aliathon

Aliathon is subject to a number of macro-economic forces which influence or may influence in the future the way in which the company operates. Aliathon sells its products in an international market place. Unfavourable exchange rates could therefore negatively impact margins and demand reviewed pricing structure for Aliathon's products. Variable rates of inflation may also directly impact Aliathon's operations.

C.5.1.3 Socio-Cultural Influences on Aliathon

Much of the increased demand for the services offered by next generation networks has been generated by socio-cultural shifts in populations throughout the world. End users desire increasingly sophisticated data-driven products and applications, and are increasingly mobile. Levels of disposable income across Asia are rapidly increasing, fuelling demand for broadband connectivity and mobile services.

Socio-Cultural influences can have a profound influence on the uptake of new applications. In South Korea for example, whose citizens represent the highest concentration of broadband users in the world, uptake of television on demand based on Internet Protocol (IPTV) has been strong. Yet in Singapore, uptake of similar services has been less emphatic; principally because societal norms (and laws) are more strictly enforced. Thus gambling and other adult multimedia, which have been mainstays of consumer demand in many regions, are less popular here. Cultural

influences and differing business practices are also likely to colour discussion between Western enterprises and emergent Asian players in NGN markets. This may affect Aliathon directly in future discussions with potential Asian clients, or indirectly through Aliathon's FPGA partners.

Security concerns have continued to resurface in the wake of terrorist attacks on and after September 11th 2001. Such concerns will influence decisions and legislation concerned with the deployment of new communications infrastructure. The extension of Mobile Location Services (MLS) and wire-tap capabilities in converged networks are two examples of technologies with security implications, and both are areas of intense activity and discussion.

C.5.1.4 Technological Influences on Aliathon

Technological factors are not merely part of the macro-environment in which Aliathon operates; rather they represent the foundation of Aliathon's business, and the single most influential external variable in determining the success of current and future operations. As such, they are dealt with in detail in the latter sections of this report. The general technological environment in which Aliathon operates is best illustrated by elaborating on the points introduced in the executive summary.

Increase in the volume and complexity of network traffic has led to the development of new technologies for more efficient networks. A key driver for network operators is the reduction in capital and operational expenditure, both the subject of increased corporate focus since the industry downturn of 2001. Such emergent technologies offer higher bandwidth capacity, greater efficiency and easier maintenance. The complexity of silicon solutions required by OEMs is increasing in response to these demands.

The Synchronous Optical Network (SONET) in North America and Japan and the Synchronous Digital Hierarchy in the rest of the world have become the established standards for optical networking. Whilst these standards continue to generate revenue at all levels in the industry, their technological bases are traditional voice telephony. Incremental increases in voice traffic through the network in recent years have been outshone by an exponential increase in data traffic. Hence the emergence of the term "converged networks" which embodies the myriad of technological advances designed to unify the transmission of voice, video and data over next generation network infrastructure.

This infrastructure is evolving from the traditional Public Switched Telephone Network (PSTN)

"circuit switched" model to a "packet switched" system, based on Internet Protocol[3](IP). IP has revolutionised data networking, but it is not optimised for transfer of delay sensitive signals such as video and voice. Techniques such as Multi-Protocol-Label-Switching (MPLS) and DiffServ have emerged to improve matters, providing Quality-of-Service (QoS) guarantees for packet based networks.

C.5.1.5 Legal Influences on Aliathon

The semiconductor industry is intensely competitive, and competitors within the industry are frequently engaged in litigation to protect intellectual property rights. Since intellectual property constitutes Aliathon's core product portfolio, legal aspects concerning protection of this portfolio are of key significance.

In addition to protecting its own intellectual property, Aliathon may need to enter into legal agreements with potential clients to indemnify the latter in the event that Aliathon inadvertently infringes an existing patent; necessitating withdrawal of the clients product. The risks associated with such infringement are mitigated by the knowledge that since FPGAs are inherently reprogrammable, reengineering of infringing designs is likely to be possible.

Additionally, potential clients may seek legal agreements to secure access to IP cores and FPGA-based systems provided by Aliathon in the event that the company enters administration. Again the nature of programmable technology mitigates the likely impact such a scenario would have on Aliathon's clients. Trade restrictions and industry regulation may also influence Aliathon's ability to operate successfully in certain markets.

C.5.2 Market Segmentation by Network Space

Aliathon operates within the communications industry, which is of vast scope. The company is not generally involved in the design or manufacture of high volume consumer products or customer premises equipment (CPE) which are largely commoditized markets with economies of huge scale.

Our target applications are high value, high complexity and principally deployed as part of the network infrastructure. In this context our markets may be broadly segmented [139] into the

[3]As distinguished from Intellectual Property

Core, Edge, Metro, Enterprise and *Access* spaces. As shown in Figure C.10 Aliathon IP is enabling leading edge capabilities across all of these network spaces.

The *core* network is the ultra high speed (typically 10Gbps or greater) optical fibre backbone carrying the aggregated traffic from all service types around and between cities, and long haul between continents. The core network is characterised by high speed and large traffic volume, with limited intelligent functionality.

The *edge* is literally the edge of the core network, comprising much of the networks intelligence in terms of traffic monitoring, policing service level agreements, aggregating different protocols, provisioning security and ensuring end-to-end quality of service.

The *metropolitan* area of the network, or *metro*, is a fibre optic infrastructure which provides high speed communications (typically 2.5Gbps or greater) over a city centre or other regional area. The *metro* is self managing within its own geographical area and provides the bridge between the *access* infrastructure and the *core*.

The *enterprise* area of the network includes equipment deployed primarily in businesses for data communications and other local area network applications. Such equipment includes switches and routers of inter-office and intra-office communications.

The *access* network encompasses the network interface to the end users, and the aggregation of end user traffic onto the *metro* infrastructure. Physical access nodes can be wired (optical fibre, twisted pair copper or HFC[4]) or wireless (wi-fi or WIMAX).

C.5.3 Market Segmentation by Technology

It is also possible to segment key markets in the communications industry according to the technologies or protocols deployed in the network. Aliathon have sourced and analysed detailed information on this basis from Infonetics Research, a leading market research firm specialising in networking and telecommunications.

The markets reviewed represent both those which are reinforcing the diversity and volume of traffic through the network, without necessarily being markets within which Aliathon would directly compete; and those which represent direct opportunities for the company. The latter were given more detailed consideration. The key points of interest are summarised here.

[4]Hybrid Fibre Coax

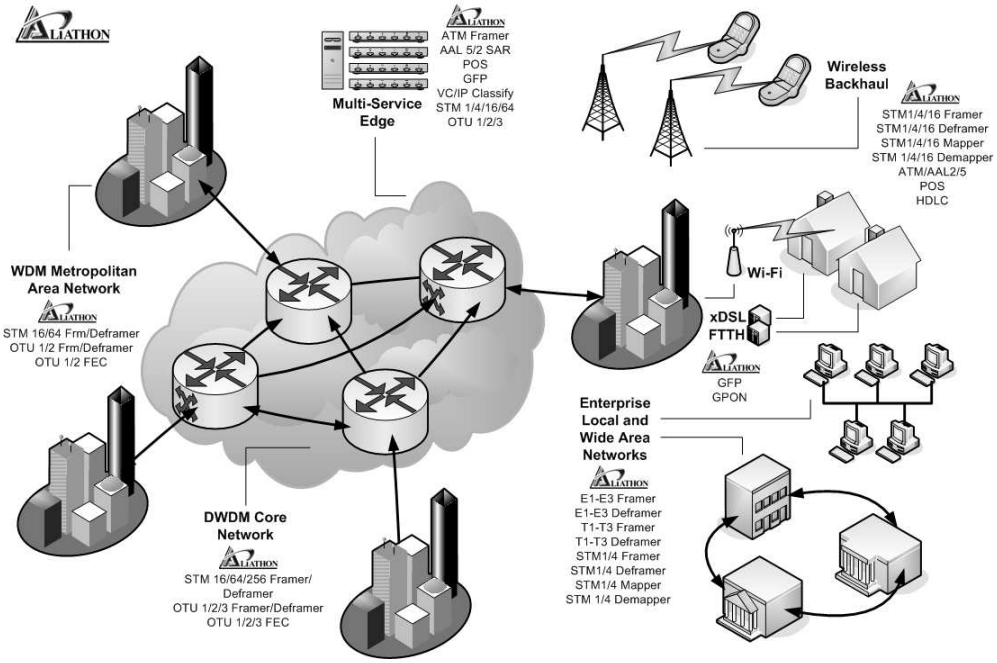

Figure C.10: *Aliathon cores across the network span*

C.5.4 Telecom/Datacom Market Fundamental Drivers

Market research indicates that the communications industry remains a vast opportunity for both the incumbent and small to medium enterprise (SME) players like Aliathon. 2003 was the last year of major cutbacks in capital expenditure, notably in the optical transport systems market where Aliathon holds significant expertise. This segment alone is poised to deliver 13% growth over the next three years [140], with revenues reaching $19.4 billion in 2008.

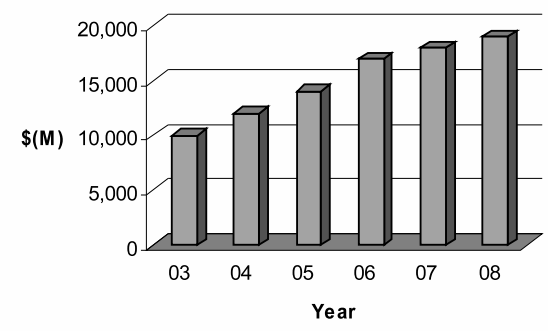

Figure C.11: *Optical Transport Systems Growth*

C.5.5 Market Analysis Highlights and Conclusions

- Fundamentally, broadband penetration is fuelling unprecedented demand for communications bandwidth. The development of next generation services, particularly video, will drive this trend further after initial broadband growth plateaus. This provides a sound economic underpinning for communications markets, and Aliathon's core business.

- In broad terms, the industry is moving slowly away from legacy Time domain Multiplexed (TDM) services to packet based networking and is currently in a hybrid/transition mode, where Multi-Service Provisioning Platforms (MSPPs) are prevalent. MSPPs can offer ATM, IP, Frame-Relay, PWE3 and Ethernet (a mix and match of legacy and emerging technologies) on the same chip. Aliathon offers industry leading solutions to support this model.

- A number of technology segments, summarised in Table C.2, offer significant growth opportunities through the forecast period ending 2008. Detailed equipment categories are given in [134]. Of particular note is strong growth in Ethernet equipment and services, and FTTx access technologies.

- SONET/SDH revenues are projected flat in both the metro and long-haul segments, but the total market size for this equipment remains extremely high. Aliathon holds significant expertise in these key technology segments.

- Ethernet is becoming a dominant transport standard, and given reported cost benefits of up to 70% at the OC-192 (10Gbps) technology node, looks set to displace legacy SONET/SDH equipment gradually over the forecast period [141]. The Operations, Administration, Maintenance and Provisioning (OAMP) aspects of Ethernet as a carrier technology are the subject of much current industry debate. Aliathon's Optical Transport Network solutions are potentially a key enabling technology for Carrier Ethernet. Our blue-chip customers agree.

- Other emerging technologies look set to drive and/or exploit the technology migration to a unified IP/MPLS network. These include IMS, GPON and PWE3. The latter technologies will be investigated as part of Aliathon's ongoing commitment to new product development, and its research partnerships with the Institute for System Level Integration.

Equipment Category	5 Yr Running Rev 2003-2008 (Millions)	5 Yr CAGR
ADSL IADs	$1,687	34%
G.SHDSL Modems/Routers	$433	20%
IP-Based DSLAMs	$6,895	30%
NG Broadband Loop Carriers	$4,999	89%
Secure Enterprise Routers	$1,503	32%
IP PBX Systems	$6,858	31%
Ethernet 1M-100M	$39,440	30%
Ethernet above 100M	$29,300	55%
FTTB	$5,737	158%
FTTH	$62	132%
RPR over Fibre	$1,436	63%
Metro SONET/SDH	$26,462	0%
Metro WDM	$10,249	22%
Long Haul SONET/SDH	$5,419	2%
Long Haul WDM	$12,570	3%

Table C.2: *Market size and forecasts in key technology segments*

C.5.6 Aliathon's Marketing Activities

As identified in C.4.3.2, Aliathon's limited marketing activity is a key area for development within the company. The companys success to date has been based on focusing significant efforts on the development of technically superior IP. As a result, non-technical aspects of the business remain underdeveloped. Competitive analysis suggests that leading ASIC and ASSP developers typically recycle 10-20% of annual revenues into sales and marketing activities. Aliathon needs to begin to scale its non-technical efforts in this direction.

The main objective is to grow Aliathon's business, by considering an appropriately scaled marketing mix [142]. Partnership with Xilinx and Altera gives Aliathon a unique opportunity as a small company to leverage market information and form close relationships with large customers. This will need to be cemented by our independent marketing efforts, which will allow us to target groups of smaller customers who may not be of strategic interest to Xilinx and Altera, but who nonetheless represent a significant opportunity for Aliathon.

C.5.6.1 Product

Fundamentally, Aliathon needs to continue to differentiate its IP based on the quality of our engineering. However, these efforts need to be complemented by excellent customer support,

Receiver	Core Price	Transmitter	Core Price
STM-4 demapper	$ 20,000.00	STM-4 mapper	$ 20,000.00
STM-4 deframer	$ 15,000.00	STM-4 framer	$ 15,000.00
E4	$ 10,000.00	E4	$ 10,000.00
E3/T3	$ 10,000.00	E3/T3	$ 10,000.00
E2/T2	$ 10,000.00	E2/T2	$ 10,000.00
E1/T1	$ 15,000.00	E1/T1	$ 15,000.00
PLCP (MC) Deframer	$ 10,000.00	PLCP (MC) Framer	$ 10,000.00
Byte-wide ATM (SC) Deframer	$ 7,000.00	Byte-wide ATM (SC) Framer	$ 7,000.00
Byte-wide ATM (MC) Deframer	$ 15,000.00	Byte-wide ATM (MC) Framer	$ 15,000.00
POS/PPP (SC) Deframer	$ 7,000.00	POS/PPP (SC) Framer	$ 7,000.00
Bit_HDLC (MC) Deframer	$ 7,000.00	Bit_HDLC (MC) Framer	$ 7,000.00
STM-16 Deframer	$ 20,000.00	STM-16 Framer	$ 20,000.00
STM-16 POS Deframer	$ 10,000.00	STM-16 POS Framer	$ 10,000.00
STM-64 Deframer	$ 30,000.00	STM-64 Framer	$ 30,000.00
OTU-1 Deframer	$ 20,000.00	OTU-1 Framer	$ 20,000.00
OTU-1 FEC Decoder	$ 40,000.00	OTU-1 FEC Encoder	$ 10,000.00
OTU-2 Deframer	$ 30,000.00	OTU-2 Framer	$ 30,000.00
OTU-2 FEC Decoder	$ 70,000.00	OTU-2 FEC Encoder	$ 20,000.00
OTU-3 Deframer	$ 50,000.00	OTU-3 Framer	$ 50,000.00
OTU-3 FEC Decoder	$ 130,000.00	OTU-3 FEC Encoder	$ 40,000.00

Table C.3: *Aliathon IP core pricing 2005*

and quality documentation for our cores, both of which are essentially part of the end product we ship to the customer. Aliathon also needs to build on relationships with new customers and our FPGA partners to better align our future product roadmap with customer needs.

C.5.6.2 Price

Aliathon's pricing strategy has not yet been formalised, but is essentially cost-based, with core prices (shown in Table C.3) based on development outlay plus profit mark up. Identifying the most appropriate price and business model for our IP sales is part of an ongoing review within Aliathon. Our current sales are based on a one-off license fee payment allowing the customer (within an agreed scope) unlimited use of the product.

This model may limit Aliathon's ability to leverage the true value of its products from customers, since there is no incremental benefit to us if the end customer users our product in high volume.

A key alternative is *cost-per-use* model, which allows Aliathon to benefit from volume sales, just as the ASIC and ASSP vendors do. Such a model is difficult to police; current FPGAs do not offer adequate protection against unlicensed reuse of IP designs.

C.5.6.3 Placement and Distribution

Aliathon is not reliant on placement or distribution of our product, since the product is easily released electronically directly to the end customer.

C.5.6.4 Promotion

Increased emphasis on promotion needs to become an integral part of Aliathon's ongoing marketing strategy. Positioning the benefits of our IP running on FPGAs is not a trivial task, and requires the education and motivation of both end customers, and the applications engineers at Xilinx and Altera. Technical presentations and training seminars represent a good forum for this. Aliathon could easily broaden its target audience for such presentations. Aliathon could also leverage current design wins as a springboard to making presentations on our product family direct to end customers

Aliathon's profile within the electronics industry as a whole is low. With adequate resources, Aliathon could raise this profile through key industry media, such as Light Reading, Lightwave or Electronics Weekly to increase awareness both of the Aliathon brand, and of the technology advantages we offer. This needs to be preceded by a review of our brand image, and an overhaul of our web presence and documentation.

Product demonstration is another cornerstone of Aliathon's promotional strategy. IP cores running on real hardware platforms are a much more powerful selling tool than simulation models or documented timing performance. Such hardware platforms can be readily connected to established industry test equipment, and the performance of Aliathon's products can be shown to meet industry standards. This dramatically boosts customer confidence in Aliathon's technical capabilities.

C.6 Strategic Plan

C.6.1 Overview

Based on the preceding analyses and the emerging growth trend in Aliathon's business outlined in financial information which follows, the company has identified some key points of strategy required to build on its successes to date. These are outlined below.

C.6.2 Migration to IP Sales

Aliathon has successfully built a portfolio of valuable intellectual property since it began trading. Initially, the income required for this portfolio development came from design services contracts with a small component from IP sales. As the portfolio has matured, the IP sales component in the revenue stream has steadily increased, such that the IP development business is now profitable and self-funding.

Consolidation and growth of Aliathon's IP sales business is thus a key strategic goal over the next two years. This will take precedence over design services activities, although the latter may continue to offer opportunities for growth and will not be discounted completely. The points which follow have been identified as essential steps towards realising this goal.

C.6.3 Increase Direct Marketing Activity

To date Aliathon has taken a reactive role in pursuing opportunities for IP sales, without a dedicated sales force or coordinated marketing strategy. There is thus significant opportunity accelerate Aliathon's growth by actively marketing and selling our products, as discussed in section C.5.6. Aliathon will also seek to leverage the support of Scottish Enterprise in this context.

C.6.4 Consolidate Partnerships with Xilinx and Altera

Aliathon's FPGA partners are of key importance. They provide leads and details for potential IP sales opportunities, and employ large sales forces which can work on Aliathon's behalf. Ongoing discussion with both key vendors will seek to identify ways in which Aliathon can realize the value of these partnerships, whether through better awareness of Aliathon's capabilities amongst Xilinx and Altera sales engineers, or access to privileged marketing information.

C.6.5 Develop Standard Products

Aliathon has the potential to greatly reduce its incremental cost of sales by developing a suite of standard products, which require minimum additional engineering effort for sales to multiple customers. This approach requires development of standard register sets and core wrappers,

which make Aliathon's products easier to reuse. Such standardisation increases the up-front development costs of Aliathon's cores, but improves profitability in the long term. Aliathon has already begun this standardisation process and will continue to integrate this into the core development process.

C.7 Financials

C.7.1 Overview

Aliathon is independently funded and privately owned, and despite the global downturn in the communications industry in 2001, has delivered year on year revenue growth and sustained profitability. As the industry returns to health worldwide, Aliathon is well positioned to accelerate its growth, and seeks the assistance of Scottish Enterprise to fulfill this potential. Aliathon emerged as an expert design services company in 2001 and has successfully migrated this business to become a leading supplier of high value communications IP cores for FPGAs.

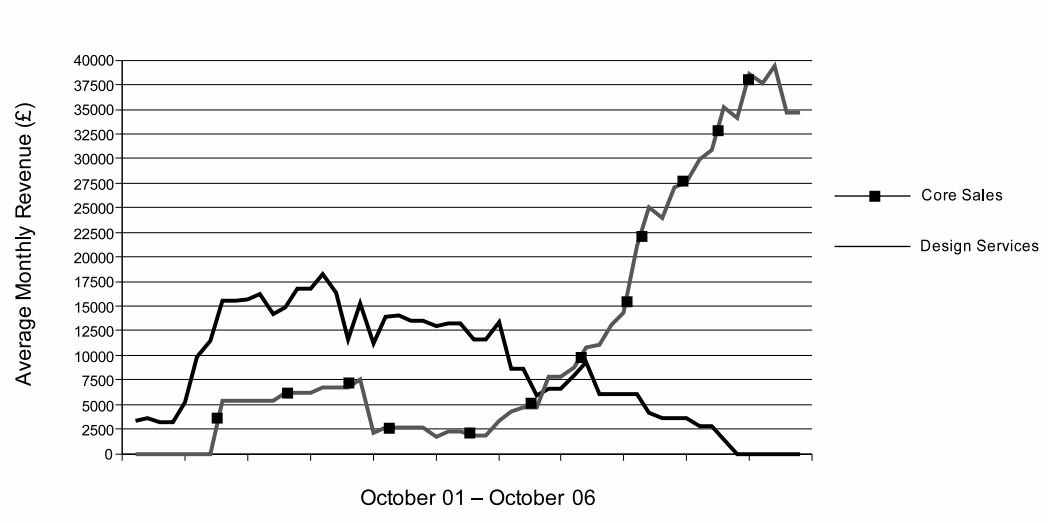

Figure C.12: *Aliathon revenue trends: IP core sales vs design services*

Ongoing contact with customers and internal estimates on the size of the potential market for Aliathon's solutions suggest that there is significant scope to consolidate and accelerate the growth trend shown in Figure C.12.

C.7.2 Revenue Projection

Based on IP core sales to date, likely future sales from current negotiations and the overall revenue opportunity identified some 12M based on an initial targeting matrix of 130 potential customers it is possible to project Aliathon's average monthly revenues through to March 2008, based on the companies current business model.

Historical data is limited, so projected figures used in cash flow and income statements are based on averaged points between a more aggressive polynomial best fit and a linear best fit line (based only on core sales since joining the Altera and Xilinx partner programmes projects) which predicts more modest growth.

Nonetheless, Aliathon's market research indicates that even the more aggressive revenue projections are achievable based on current resources, for a number of reasons. Firstly, Aliathon's partnerships with Xilinx and Altera are still in their infancy. Aliathon can continue to leverage high quality support from these vendors to grow revenues.

Additionally, Aliathon's current business is based on minimal sales and marketing activities. Development of the latter represents a significant opportunity for revenue growth. Aliathon's incremental cost of sales is decreasing with our increased focus on standard products. Additionally, Aliathon need only convert 6% of the total opportunities identified in C.3.2.2, to outperform the per-annum revenue projection for 2008.

Furthermore, continuing product development will increase Aliathon's total market opportunity. Aliathon's current market size estimates are based on targeting customers with our existing product portfolio, which is strongest in the time division multiplexed (TDM) domain.

Aliathon's market research indicates that the industry is shifting towards packet-based technologies such as IP/MPLS with Ethernet as the carrier standard. Aliathon's expertise will allow us to develop leading edge products targeting these emergent technologies; research work on packet classification for high speed internet routing engines had already commenced. With these additional capabilities in our portfolio, Aliathon can actively target a new set of potential customers who are offering network processing, IP/MPLS and Carrier Ethernet equipment.

Month	Optimistic Projection (£)	Pessimistic Projection (£)	Mean (£)
Apr-06	45935	39267	42601
Jul-06	55057	43045	49051
Oct-06	64835	46823	55829
Jan-07	75135	50601	62868
Apr-07	85802	54379	70091
Jul-07	96653	58158	77405
Oct-07	107484	61936	84710
Jan-08	118066	65714	91890

Table C.4: *Aliathon's projected revenues by selected quarter*

C.8 Supporting Information

C.8.1 Background to the Aliathon Portfolio

Aliathon's products are used in a wide variety of communications and networking applications. Historically, these communications networks have been based on voice telephony channels, running at 64Kbits/second, each capable of faithfully transmitting the information required to hold a single telephone conversation. These channels could then be grouped or "multiplexed" together to form higher bandwidth connections between network elements, with each connection supporting multiple calls.

In the early days, different networks based on this multiplexing idea ran at approximately equal speeds, in what became known as the Plesiochronous Digital Hierarchy (PDH). As the number of users and the complexity of PDH networks grew, the shortfalls and limitations of the technology became apparent. In particular, it became increasingly difficult and expensive to extract individual calls from the higher bandwidth connections. A lack of standardisation made interoperation of equipment between different manufacturers a hit and miss affair, and network management functions were limited.

Enter the Synchronous Digital Hierarchy (SDH) and the Synchronous Optical Network (SONET). These standards facilitated interoperation of network equipment across the globe, and provided the management functions required for very reliable communication between increasingly complex network elements. These elements are now typically connected by optical fibre at speeds up to 40Gbps; thats 625,000 phone calls down a single fibre connection.

This migration to high speed fibre connectivity has been consolidated further with the introduction of the new Optical Transport Network (OTN) standard ITU-T G.709. This technology is optimised for the management of optical channels, and the correction of transmission errors. Such standards will provide the backbone infrastructure for ongoing bandwidth and revenue growth in the communications industry. Aliathon's Products in Todays Networks

As networks grow in size and complexity, the ways in which lower speed connections are combined to form higher capacity links become commensurately more complex. The number of possible combinations and permutations is enormous. Aliathon IP is designed to cover all these possibilities; supporting legacy PDH rates, SONET/SDH and the emerging OTN standards, all of which co-exist in todays networks.

In parallel with this high level growth trend towards greater bandwidth is an industry movement towards convergence. Demand for traditional voice telephony has increased incrementally in recent years, but demand for data services has exploded. Thus, network equipment manufacturers (NEMs) have sought ways to improve the datacoms capabilities of their existing telecoms networks. This has led to the so-called Next Generation SONET (NG-SONET) services , which are already being enabled by Aliathon IP.

This convergence model looks set to continue apace, with the ultimate goal of voice, video and data delivery over a single, unified network the Internet. Aliathon's product portfolio and roadmap will track these trends closely, and enable our customers to rapidly generate revenue from the latest available technologies.

C.8.2 IP Taxonomy

C.8.2.1 Telecoms, Functional Blocks

- **Mappers**: Perform the mapping of lower rate signals into higher capacity links; for example mapping PDH signals into the SONET/SDH domain. This involves careful control of network timing and management overhead.
- **Framers**: Format data according to a standardised frame structure to ensure reliable interoperability of equipment.
- **FEC**: Forward Error Correction. A technique for encoding and decoding data using

standardised schemes to improve the reliability of communications over optical fibre.

- **VCAT**: Virtual Concatenation. A technique for improving the improving the efficiency of bandwidth utilisation in SONET/SDH systems, for payload data which does not fit conveniently into traditional data containers.

C.8.2.2 Transmission Rates

- **E1**: Standard European E-Carrier transmission rate of 2.048Mbps.
- **E3**: Standard European E-Carrier transmission rate of 34.368Mbps.
- **T1**: Standard North American T-Carrier transmission rate of 1.544Mbps.
- **T3**: Standard North American T-Carrier transmission rate of 44.736Mbps.
- **STM-1**: Standard SDH3 transmission rate of 155.52Mbps.
- **STM-4**: Standard SDH transmission rate of 622.08Mbps.
- **STM-16**: Standard SDH transmission rate of 2.488Gbps.
- **STM-64**: Standard SDH transmission rate of 9.953Gbps.
- **STM-256**: Standard SDH transmission rate of 39.813Gbps.

C.8.2.3 Datacoms, Network Protocols

- **ATM**: Asynchronous Transfer Mode. A high speed network protocol with short packet length to reduce transit delays; optimised for real time voice and video traffic.
- **AAL5/2**: ATM Adaptation Layers. Protocols for adapting multi-cell higher layer data units into ATM.
- **POS**: Packet over SONET. A technique for mapping packetised traffic such as IP or Ethernet into the time multiplexed SONET domain.
- **HDLC**: High Level Data Link Control. An ISO communications protocol used in packet switched networks.

- **GFP**: Generic Framing Procedure. A technique for mapping packetised traffic into the time multiplexed domain.
- **VC/IP Classification**: Techniques for identifying specific packet types in the IP/MPLS4 network for higher speed, lower power routing.

C.8.3 Target Customers

A summary of Aliathon's potential clients, based on an a review of activity in key communications market segments is shown in Figures C.13 and C.14.

Networking Enterprise Mid-Low Range	3Com 5000
Access- Transmission Access Media Gateway	Aastra CVX 600, CVX 1800
Access- Transmission Access DSLAM	AFC Telliant
Access- Transmission Access FTTx -PON	Alloptic GigaForce edgeGEAR2000
Access- Transmission Access PBX -IP -PBX	Altigen Altiserv
Access- Transmission Access CMTS	Arris C3, C4, 1500,Cornerstone (1000, 1500)
Networking Enterprise L2-L3 Switches	Avaya P33x, P130, C460
Access- Transmission Access CMTS	BigBand Cuda 12000
Access- Transmission Transmission DWDM (LH)	Calient DiamondWave
Access- Transmission Access DSLAM	C-COM SmartDSLAM
Access- Transmission Access Media Gateway	Cedar Point SAFARI C3
Networking Enterprise L4-7 Switches	Celestix FVxxx
Networking Public Network Core Routers	Chiaro Enstara
Access- Transmission Access Media Gateway	Cirpack Cirpack TN
Access- Transmission Access DSLAM	Copper Mountain CopperEdge, VantEdge
Access- Transmission Access Media Gateway	CopperCom CSX1100-2100, CopperController
Networking Enterprise L4-7 Switches	CyberGuard LX
Networking Enterprise L2-L3 Switches	D-Link DSS, DES, DGS
Networking Enterprise High-End Routers	Enterasys 8000, 8600
Networking Enterprise L2-L3 Switches	Extreme Summit 24e2, BlackDiamond 6800
Networking Enterprise L4-7 Switches	F5 BigIP 5000, BigIP 2000, BigIP 1000
Access- Transmission Access FTTx -PON	FlexLight Optimate 1000LT, 2500LT
Networking Enterprise L2-L3 Switches	Force10 E-Series
Networking Public Network Edge Router	Foundry NetIron (400, 800, 1500)
Networking Enterprise L2-L3 Switches	HP ProCurve
Access- Transmission Access FTTx -PON	iamba XL100B, XL200B
Access- Transmission Transmission DWDM (Metro)	Internet Photonics LightStack, LightHandler
Networking Enterprise L4-7 Switches	Intrusion SecureNet 7000, 5000, 2000, 2600, 4000
Networking Enterprise L4-7 Switches	iPolicy ipEnforcer
Access- Transmission Access FTTx -PON	LGE StarDLC-6400
Access- Transmission Access PBX -IP -PBX	Mitel 3x00 ICP
Access- Transmission Access FTTx -PON	Motorola QC3000, QB5000
Access- Transmission Transmission CO Switch	NetCentrex CCS
Networking Enterprise L2-L3 Switches	NetGear GS Series, GSM712
Networking Enterprise L4-7 Switches	NFR Security NID
Networking Enterprise L4-7 Switches	Niksun NetDetector
Access- Transmission Access Media Gateway	Nuera Nuera BTX-8-21, GX-8-21
Access- Transmission Access DLC	Occam BLC 1100, 1200, 1240, 6000
Access- Transmission Access FTTx -PON	Optical Solutions FiberPath 400
Networking Enterprise L4-7 Switches	Radware WSD, CID, CSD, LinkProof, FireProof
Networking Public Network Edge Router	Riverstone RS 8000, RS 8600,RS 38000
Access- Transmission Access Media Gateway	sentitO IVG Intelligent Voice Gateway
Networking Enterprise L2-L3 Switches	SMC TigerSwitch
Networking Enterprise L4-7 Switches	Sourcefire Network Sensor
Access- Transmission Access PBX -IP -PBX	Sphere Spherical Manager
Access- Transmission Access PBX -IP -PBX	Swyx SwyxServer
Networking Enterprise L4-7 Switches	Symantec ManHunt, ManTrap
Access- Transmission Access FTTx -PON	Terawave TW-300, TW-400 ONT
Networking Enterprise L4-7 Switches	TippingPoint UnityOne 2000, 600
Networking Enterprise L4-7 Switches	TopLayer Attack Mitigator IPS
Networking Enterprise High-End Routers	Vanguard 7300
Access- Transmission Access Media Gateway	Veraz I-Gate 4000-4000 PRO
Access- Transmission Access PBX -IP -PBX	Vertical Networks InstantOffice
Access- Transmission Access DSLAM	WaiLAN DeltaFire500
Access- Transmission Access FTTx -PON	Wave7 Last Mile Core
Access- Transmission Access DSLAM	ZyXel IES-1000-2000-3000, AES-1000
Access- Transmission Transmission PDH Line Card	Brooktrout TR1000, TR2000
Access- Transmission Access FTTx -PON	Nayna ExpressSTREAM
Access- Transmission Access FTTx -PON	TelStrat Inteleflex BLC
Access- Transmission Access PBX -IP -PBX	Zultys MX1200, MX250

Figure C.13: *Target Customer Matrix - I*

Access- Transmission Access Media Gateway	Convergent ICS2000
Access- Transmission Access DSLAM	Corecess 6800, 8100, DSLinX
Access- Transmission Test Test Equipment	Consultronics Puma 4000
Access- Transmission Access Media Gateway	MetaSwitch VP3500
Access- Transmission Transmission SONET-SDH	Appian OSAP
Access- Transmission Access DSLAM	Avail Networks Frontera (2000, 4000)
Access- Transmission Access DSLAM	Paradyne 4200
Networking Enterprise High-End Routers	Memotec CX950, CX900e, CX1000e
Access- Transmission Access DSLAM	Loop Telecom Loop-H 3780-U1000
Networking Public Network Edge Router	Quarry iQ4000, iQ8000
Access- Transmission Transmission Protocol Conv	Carriercom N/A
Access- Transmission Transmission CO Switch	Italtel iMSS Softswitch
Access- Transmission Access DLC	Keymile UMUX 1200, 1500
Networking Public Network Wireless Analyser	Nethawk Nethawk M5, 2G 3G
Access- Transmission Transmission Monitoring	Resi MNM
Access- Transmission Access Media Gateway	Tekelec 7000 Class 5 Packet Switch
Access- Transmission Access DSLAM	Samsung AceMAP, AceLink
Access- Transmission Access Media Gateway	Sonus GSX9000
Access- Transmission Access Media Gateway	Clarent BHG 1000-8000
Access- Transmission Transmission DXC	Orion VCL-MegaConnect
Access- Transmission Access DSLAM	Adtran TA(1100, 1200, 3000)
Access- Transmission Test Test Equipment	Circadient OST-10D, OST-10M
Access- Transmission Transmission SONET-SDH	Redback SmartEdge 800
Access- Transmission Access FTTxPON	Carrier Access Exxtenz-B-ONT
Access- Transmission Transmission SONET-SDH	Apcon Intellapatch Blade/Chassis
Access- Transmission Access FTTxPON	Calix C7 BPON
Access- Transmission Access FTTxPON	Entrisphere BLM 1500 OLT
Access- Transmission Transmission DWDM(Metro)	Lumentis Mentis
Networking Public Network Edge Router	Laurel Networks ST200
Networking Public Network Edge Router	net.com Scream 50, Scream 100
Access- Transmission Transmission Optical Switch	Meriton 7200 OADX
Networking Enterprise High-End Routers	ImageStream Gateway, Rebel Pro
Access- Transmission Transmission DWDM(Metro)	LuxN WavPortal, WavFarer, WavStation
Access- Transmission Transmission MS WAN Switch	Ceterus UTX8212, UTSxxxx
Networking Public Network Core Routers	Avici QSR, SSR, TSR
Access- Transmission Transmission DWDM(LH)	Corvis CorWave LR, XL-XF, ON, OCS
Access- Transmission Transmission SONET-SDH	Net Insight Nimbra 210, 220, 290, ONE
Access- Transmission Access DSLAM	Allied Telesyn 7400, 7700
Access- Transmission Transmission SONET-SDH	Sycamore SN 3000, 16000 SC
Access- Transmission Transmission DWDM, DXC	White Rock VLX, ETS, OTS
Access- Transmission Transmission SONET-SDH	Corrigent CM-100
Access- Transmission Transmission SONET-SDH	Turin 1600, 2000
Networking Public Network Monitoring	Endace DAG
Access- Transmission Transmission SONET-SDH	ECI SDM (-1c, -1, -4, -16)
Access- Transmission Access DLC	ADC Avidia 2200
Access- Transmission Transmission DWDM(Metro)	ADVA FSP (1, 500, 1000, 2000, 3000)
Access- Transmission Transmission DWDM(Metro)	Movaz RAYstar, RAYexpress
Access- Transmission Transmission SONET-SDH	NEC SMS-150, SMS-600
Access- Transmission Test Test Equipment	Acterna ONTxxx, ANTxxx
Access- Transmission Transmission SONET-SDH	Alcatel 1641, 1651 , 1661, 1632
Access- Transmission Test Test Equipment	Anritsu MD, MP
Access- Transmission Transmission SONET-SDH	Ciena ONWAVE SONET-SDH
Access- Transmission Transmission SONET-SDH	Cisco 15302, 15305
Access- Transmission Test Test Equipment	Digital Lightwave OTS, NICs
Access- Transmission Transmission SONET-SDH	Ericsson AXD155-620,SMA-16
Access- Transmission Transmission SONET-SDH	Fujitsu FLASHWAVE 2540,FLx
Access- Transmission Access DSLAM	Harbour Networks Hammer10000
Access- Transmission Access FTTxPON	Hitachi AMN1200, 1210
Access- Transmission Transmission SONET-SDH	Huawei OptiX 155-622
Networking Public Network Edge Router	Juniper ERX-3xx, 7xx, 14xx, M5-M320
Access- Transmission Transmission SONET-SDH	Lucent WaveStar AM1, ADMx, DDM/FT-2000
Access- Transmission Transmission SONET-SDH	Marconi SMA1, SMA4, SMA16,MSH11, MSH41
Access- Transmission Access DSLAM	Nokia D50e, D500
Access- Transmission Transmission SONET-SDH	Nortel OPTera Connect DX,TN-1, TN-P
Access- Transmission Transmission SONET-SDH	Siemens SLDx,SMAx, SURPASS hiT
Access- Transmission Test Test Equipment	Spirent AX, AE
Access- Transmission Transmission SONET-SDH	Tellabs 6310
Access- Transmission Access FTTxPON	UTStarcom BBS1000 GEPON
Access- Transmission Transmission SONET-SDH	Zhone TeraMatrix

Figure C.14: *Target Customer Matrix - II*

References

[1] G. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38 No.8 April, 1965.

[2] C. Edwards, "The Many Lives of Moore's Law," *IET Engineering and Technology*, vol. 3, 2008.

[3] "International Technology Roadmap for Semiconductors." 2005 edition, Available at `http://www.itrs.net/Links/2005ITRS/Home2005.htm`.

[4] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-On-A-Chip Designs*. Springer N.Y., 1998.

[5] K. Werner, "VSI Alliance Quality IP Metric." Design and Reuse., available at `http://www.us.design-reuse.com/articles/7182/vsi-alliance-quality-ip-metric.html`.

[6] D. Robinson, "Shorten and Simplify SoC Verification Using a Generic eVC." Verilab Ltd., available at `http://www.verilab.com/download.htm`, 2005.

[7] H. Chang, L. Cooke, and M. Hunt, *Surviving the SoC Revolution: A guide to Platform-Based Design*. Springer N.Y., 1999.

[8] J. Blyler, "Navigating the Silicon Jungle: FPGA or ASIC?," *Chip Design Magazine*, June/July 2005.

[9] J. Plofsky, "The Changing Economics of FPGAs, ASICs and ASSPs," *Real Time Computing*, April 2003.

[10] T. Murgan, M. Petrov, M. Majer, P. Zipf, M. Glesner, U. Heinkel, J. Pleickhardt, and B. Bleisteiner, "Adaptive Architectures for an OTN Processor: Reducing Design Costs through Reconfigurability and Multiprocessing," in *ACM Conference on Computing Frontiers*, pp. 404–418, 2004.

[11] "Interfaces for the Optical Transport Network (OTN)." ITU-T Recommendation G.709, February 2001.

[12] T. K. Truong, J. H. Jeng, and T. C. Cheng, "A New Decoding Algorithm for Correcting both Erasures and Errors of Reed-Solomon Codes," in *IEEE Transactions on Communications*, pp. 381–388, 2003.

[13] D. V. Sarwate and N. R. Shanbhag, "High-speed Architectures for Reed-Solomon Decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 5, pp. 641–655, 2001.

[14] "Pseudowire Emulation Edge-to-Edge (PWE3) Charter." Available at `http://www.ietf.org/html.charters/pwe3-charter.html`.

[15] B. Vöcking, "How Asymmetry Helps Load Balancing," in *40th IEEE Symposium on Foundations of Computer Science*, pp. 131–141, 1999.

[16] A. Broder and M. Mitzenmacher, "Using Multiple Hash Functions to Improve IP Lookups," in *IEEE Infocom*, 2001.

[17] "Network Node Interface for the Synchronous Digital Hierarchy." ITU-T Recommendation G.707, November 1988.

[18] "Navigating the Metro Protocol Maze: Next Generation SONET/SDH Networks." Intel Corporation, available at `http://whitepapers.zdnet.com/whitepaper.aspx?docid=97161`, 2004.

[19] P. Richardson, "G.709 and the Optical Transport Network (OTN) Interface : Understanding the Logic Behind this New Standard." Digital Lightwave, available at `http://www.lightwave.com/dirSupport/whitepapers.aspx`, 2003.

[20] M. Howard, "Metro Ethernet Equipment Biannual Worldwide Market Size and Forecasts." Infonetics Research Market Report, March 2005.

[21] M. Howard, "DSL Aggregation Hardware Quarterly Worldwide Market Share and Forecasts for 1Q05." Infonetics Research Market Report, May 2005.

[22] M. Howard, "Optical Network Hardware Quarterly Worldwide Market Share and Forecasts for 1Q05." Infonetics Research Market Report, May 2005.

[23] R. Dearborn, L. Johnston, K. Mitchell, and L. Whitcomb, "Service Provider Plans for IP, MPLS, and ATM, North America and Europe 2004." Infonetics Research Market Report, December 2004.

[24] *Virtex II Pro Platform FPGA Handbook*. Xilinx Inc., UG012 (v1.0), 2002.

[25] G. C. Ahlquist, B. Nelson, and M. Rice, "Optimal Finite Field Multipliers for FPGAs," in *9th International Workshop on Field-Programmable Logic and Applications*, 1999.

[26] I. Stamoúlis, N. Ford, G. Dunnett, M. White, and L. P.F., "VHDL Methodologies for Effective Implementation on FPGA Devices and Subsequent Transition to ASIC Technology," in *Design Automation and Test in Europre (DATE)*, 1998.

[27] B. Falkowski, "Equivalence Checking for Digital Circuits," *IEEE Potentials*, vol. 23, pp. 21 – 23, April-May 2004.

[28] S. Vasudevan, V. Viswanath, J. Abraham, and J. Tu, "Automatic Decomposition for Sequential Equivalence Checking of System Level and RTL Descriptions," in *4th IEEE/ACM International Conference on Formal Methods and Models for Co-Design*, pp. 71–80, 2006.

[29] D. Hegarty and S. McDonald, "An FPGA-based Configurable Network Interface Card," in *IEEE International Conference on Systems (ICONS)*, April 2006.

[30] W. Ford and M. Baum, *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption*. Prentice-Hall N.J., 2nd ed., 2000.

[31] D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," in *Proceedings of the Institute of Radio Engineers (IRE)*, pp. 1098–1101, 1952.

[32] S. Lin and J. Daniel J. Costello, *Error Control Coding: Fundamentals and Applications*. Prentice-Hall Computer Applications in Electrical Engineering, Prentice-Hall, 1983.

[33] R. E. Blahut, *Algebraic Codes for Data Transmission*. Cambridge University Press, 1st ed., 2003.

[34] I. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *Journal of the Society of Industrial and Applied Mathematics*, vol. 8, pp. 300–304, 1960.

[35] R. Hill, *A First Course in Coding Theory*. Oxford Applied Mathematics and Computing Science Series, Clarendon Press, 1st ed., 1986.

[36] O. Pretzel, *Error-Correcting Codes and Finite Fields*. Oxford Applied Mathematics and Computing Science Series, Clarendon Press, 1st ed., 1992.

[37] R. Hamming, "Error Detecting and Error Correcting Codes," *The Bell System Technical Journal*, vol. 29, pp. 147–160, 1950.

[38] R. Bose and D. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *IEEE Transactions on Information Theory and Control*, vol. 3, pp. 68–79, 1960.

[39] A. Hocquenghem, "Codes Correcteurs d'Erreurs," *Chiffres*, vol. 2, pp. 147–156, 1959.

[40] *Stratix Device Handbook.* vers.3.3 vol 1, Altera Corp., 2005.

[41] W. W. Peterson and E. Weldon, *Error Correcting Codes*. The MIT Press, 2nd ed., 1972.

[42] E. D. Mastrovito, "On Fast Galois-field Multiplication," in *IEEE International Symposium on Information Theory*, pp. 348–348, 1991.

[43] C. Paar, P. Fleischmann, and P. Roeise, "Efficient Multiplier Architectures for Galois Fields GF$(2^4)^n$," *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 162–170, 1998.

[44] G. C. Ahlquist, B. E. Nelson, and M. D. Rice, "Synthesis of Small and Fast Finite Field Multipliers for Field Programmable Gate Arrays." Available at `http://www.cambr.uidaho.edu/symposiums/symp11/Alquist_NASA_2003_main.pdf`, 2003.

[45] M. Goel and N. R. Shanbhag, "Low-power Channel Coding via Dynamic Reconfiguration," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP.*, vol. 4, pp. 1893–1896 vol.4, 1999.

[46] M. A. Hasan and V. K. Bhargava, "Architecture for a Low Complexity Rate-adaptive Reed-Solomon Encoder," *IEEE Transactions on Computers*, vol. 44, no. 7, pp. 938–942, 1995.

[47] L. Fredrickson, "Word-wise Processing for Reed-Solomon Codes." United States Patent No. 5,757,826, May 1998.

[48] "Finite Field Calculator and Reed Solomon Simulator." Bell Laboratories, available at `http://www.cm.bell-labs.com/~who/emina/applets/FFCalc.html`.

[49] E. R. Berlekamp, *Algebraic Coding Theory*. Aegean Park Press, 2nd ed., 1984.

[50] W. Peterson, "Encoding and Error-correction Procedures for the Bose-Chaudhuri Codes," *IEEE Transactions on Information Theory*, vol. 6, no. 4, pp. 459–470, 1960.

[51] R. Chien, "Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghem Codes in pm Symbols," *Journal of the Society of Industrial and Applied Mathematics*, vol. 8, pp. 300–304, 1960.

[52] G. Forney, "On Decoding BCH Codes," *IEEE Transactions on Information Theory*, vol. IT-11, pp. 549–557, 1965.

[53] B.-Z. Shen, L.-J. Weng, and D. L. Langer, "Parallel Input Output Combined System for Producing Error Correction Code Redundancy Symbols and Error Syndromes." United States Patent No. 6,493,845, 2002.

[54] K. Seki, K. Mikami, M. Baba, N. Shinohara, S. Suzuki, H. Tezuka, S. Uchino, N. Okada, Y. Kakinuma, and A. Katayama, "Single-chip 10.7 Gb/s FEC Codec LSI Using Time-multiplexed RS Decoder," in *IEEE Conference on Custom Integrated Circuits*, pp. 289–292, 2001.

[55] A. G. M. Strollo, N. Petra, D. De Caro, and E. Napoli, "An Area-efficient High-speed Reed-Solomon Decoder in 0.25 Micron CMOS," in *European Solid-State Circuits Conference (ESSCIRC)*, pp. 479–482, 2004.

[56] J. Massey, "Shift-register Synthesis and BCH Decoding," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, 1969.

[57] H. Burton, "Inversionless Decoding of Binary BCH Codes," *IEEE Transactions on Information Theory*, vol. 17, no. 4, pp. 464–466, 1971.

[58] J.-H. Jeng and T.-K. Truong, "On Decoding of Both Errors and Erasures of a Reed-Solomon Code Using an Inverse-free Berlekamp-Massey Algorithm," *IEEE Transactions on Communications*, vol. 47, no. 10, pp. 1488–1494, 1999.

[59] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "An Erasures-and-Errors Decoding Algorithm for Goppa Codes," *IEEE Transactions on Information Theory*, vol. 22, no. 2, pp. 238–241, 1976.

[60] Y. W. Chang, J. H. Jeng, and T. K. Truong, "VLSI Architecture Design of Modified Euclidean Algorithm for Reed-Solomon Code," pp. 304–306, 2003.

[61] T. K. Truong, S.-L. Fu, and T. C. Cheng, "Reed-Solomon Decoder and VLSI Implementation Thereof." United States Patent No. 6,209,115, 2001.

[62] S. Iyer, R. Kompella, and A. Shelat, "ClassiPI: An Architecture for Fast and Flexible Packet Classification," *IEEE Network Special Issue, 2001*, vol. 15, no. 2, 2001.

[63] A. Stephenson, "Diffserv and MPLS: A Quality Choice," *Data Communications International*, vol. 27, pp. 73–77, November 1998.

[64] B. Caswell and J. Hewlett, "Snort Users Manual 2.6.0," May 2006. Available at `http://www.mirrors.wiretapped.net/security/network-intrusion-detection/snort/snort-MANUAL.pdf`.

[65] "Internetworking Operating System (IOS) Netflow." Cisco Systems, available at `http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html`.

[66] "Common Control and Measurment Plane Charter (CCAMP)." IETF, available at `http://www.ietf.org/html.charters/ccamp-charter.html`.

[67] G. Huston, "Analyzing the Internet BGP Routing Table," Tech. Rep. 1, The Internet Protocol Journal, March 2001.

[68] F. Baboescu, S. Sumeet, and G. Varghese, "Packet Classification for Core Routers: Is there an Alternative to CAMs?," in *INFOCOM*, vol. 1, pp. 53–63 vol.1, 2003.

[69] "NetEnforcer Product Overview." Allot Communications, available at `http://www.allot.com/products/ACfamily_DS.htm`.

[70] K. Lan, A. Hussain, and D. Dutta, "Effect of Malicious Traffic on the Network," in *Passive and Active Measurement Workshop (PAM)*, 2003.

[71] "Understanding IP Addressing: Everything You Ever Wanted to Know." 3COM Corporation, available at `http://www.3com.com/other/pdfs/infra/corpinfo/en_US/501302.pdf`, 2001.

[72] "Applicability Statement for the Implementation of Classless Inter-Domain Routing (CIDR)." IETF Request for Comments 1517, available at `http://www.ietf.org/rfc/rfc1517.txt`.

[73] T. Lakshman and D. Stidialis, "High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching," in *SIGCOMM*, 1998.

[74] P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE Network*, 2001.

[75] R. Montoye, "Apparatus for Storing "Don't Care" in a Content Addressable Memory Cell." United States Patent No. 5,319,590, 1994.

[76] D. E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," Tech. Rep. WUCSE-2004-24, Washington University in Saint Louis, 2004.

[77] S. Antonanos, K. Anagnostakis, and E. Markatos, "Generating Realistic Workloads for Network Intrusion Detection Systems," in *ACM Workshop on Software and Performance*, ACM, 2004.

[78] D. Knuth, J. Morris, and V. Pratt, "Fast Pattern Matching in Strings," Tech. Rep. CS-74-440, Stanford University, 1974.

[79] A. Aho and M. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," in *Comunications of the ACM*, vol. 18, 1975.

[80] R. Boyer and J. Moore, "A Fast String Searching Algorithm," in *Communications of the Association for Computing Machinery*, pp. 762–772, 1977.

[81] S. Wu and U. Manber, "A Fast Algorithm for Multi-Pattern Searching," Tech. Rep. TR-94-17, University of Arizona, 1994.

[82] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, "Deterministic Memory-efficient String Matching Algorithms for Intrusion Detection," tech. rep., IEEE INFOCOMM, 2004. 2628-2639 vol.4.

[83] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small Forwarding Tables for Fast Routing Lookups," in *SIGCOMM*, 1997.

[84] W. Eatherton, Z. Dittia, and G. Varghese, "Tree Bitmap: Hardware/Software IP Lookups with Incremental Updates," *ACM Computer Communications Review*, vol. 34, 2004.

[85] C. Clark and D. Schimmel, "Scalable Pattern Matching for High Speed Networks," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2004.

[86] P. Gupta and N. McKeown, "Packet Classification using Hierarchical Intelligent Cuttings," *Hot Interconnects*, 1999.

[87] V.Srinivasan, G.Varghese, S.Suri, and M.Waldvogel, "Fast and Scalable Layer Four Switching," *ACM SIGCOMM Computer Communication Review*, 1998.

[88] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," in *SIGCOMM*, (Cambridge, MA), pp. 147–160, 1999.

[89] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner, "Router Plugins: A Software Architecture for Next-generation Routers," *Networking, IEEE/ACM Transactions on*, vol. 8, no. 1, pp. 2–15, 2000. 1063-6692.

[90] F. Baboescu and G. Varghese, "Scalable Packet Classification," *Networking, IEEE/ACM Transactions on*, vol. 13, no. 1, pp. 2–14, 2005.

[91] V. Srinivasan and G. Varghese, "Faster IP Lookups using Controlled Prefix Expansion," in *ACM SIGMETRICS*, 1998.

[92] "Carrier Router System CRS-1 Datasheet." Cisco Systems, available at `http://www.cisco.com/en/US/products/ps5763/prod_literature.html`.

[93] V. Bennett, "Method for Performing Optimized Intelligent Searches of Knowledge Bases Using Submaps Associated with Search Object." United States Patent No. 5,813,001, 1998.

[94] A. Tal and G. Itzchak, "Look-ahead Tree Structure." United States Patent No. 6,532,457, 2003.

[95] D. Knuth, *The Art of Computer Programming*, vol. 3. Addison-Wesley, 1998.

[96] D. Taylor, A. Chandra, Y. Chen, S. Dharmapurikar, J. Lockwood, W. Tang, and J. Turner, "System-on-Chip Packet Processor for an Experimental Network Services Platform," in *Proceedings of IEEE Globecomm*, 2003.

[97] M. Raab and A. Steger, "Balls into Bins - A Simple and Tight Analysis," in *Second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, October 1998.

[98] G. Gonnet, "Expected Length of the Longest Probe Sequence in Hash Code Searching," in *Journal of the ACM*, vol. 28, pp. 289–304, April 1981.

[99] P. Larson, "Analysis of Uniform Hashing," *Journal of the ACM*, vol. 30, pp. 805–819, October 1983.

[100] "MD5 Processor Core Users Manual." Ocean Logic Pty Ltd, available at `http://www.ocean-logic.com/pub/OL_MD5.pdf`.

[101] A. Broder and A. Karlin, "Multilevel Adaptive Hashing," in *1st ACM-SIAM Symposium on Discrete Algorithms*, 1990.

[102] H. Lim, S. Ji-Hyun, and J. Yeo-Jin, "High Speed IP Address Lookup Architecture Using Hashing," *IEEE Communications Letters*, vol. 7, October 2003.

[103] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–425, 1970.

[104] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep Packet Inspection using Parallel Bloom Filters," *IEEE Micro*, vol. 24, pp. 52–61, 2004.

[105] S. Dharmapurikar, P. Krishnamurthy, and D. E.Taylor, "Longest Prefix Matching using Bloom Filters," in *SIGCOMM*, (Karlsruhe, Germany), ACM, 2003.

[106] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing," in *SIGCOMM*, August 2005.

[107] Y. Azar, A. Broder, A. Karlin, and E. Upfal, "Balanced Allocations," in *26th ACM Symposium on the Theory of Computing*, pp. 593–602, 1994.

[108] R. Cole, B. Maggs, F. Meyer auf der Heide, M. Mitzenmacher, A. Richa, K. Schröder, R. Sitaraman, and B. Vöcking, "Randomized Protocols for Low-Congestion Circuit Routing in Multistage Interconnection Networks," in *30th Annual ACM Symposium on Theory of Computing*, pp. 378–388, 1998.

[109] R. Cole, A. Frieze, B. Maggs, M. Mitzenmacher, A. Richa, R. Sitaraman, and E. Upfal, "On Balls and Bins with Deletions," in *2nd. International Workshop on Randomization and Approximation Techniques in Computer Science*, pp. 145–158, 1998.

[110] M. Mitzenmacher, *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, University of California, Berkeley, 1996.

[111] M. Mitzenmacher, "Studying Balanced Allocations with Differential Equations," Tech. Rep. 024, Systems Research Center, October 1997.

[112] M. Mitzenmacher and B. Vöcking, "The Asymptotics of Selecting the Shortest of Two, Improved," in *37th Allerton Conf. on Communication, Control, and Computing*, 1999.

[113] W. Eatherton and Z. Dittia, "Data Structure Using a Tree Bitmap and Method for Rapid Classification of Data in a Database," 2003. United States Patent No. 6,560,610.

[114] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Method and Apparatus for Detecting Predefined Signatures in Packet Payload Using Bloom Filters." United States Patent Application No. 20050086520, 2005.

[115] S. Dharmapurikar, P. Krishnamurthy, and D. Taylor, "Method and System for Performing Longest Prefix Matching for Network Address Lookup using Bloom Filters." United States Patent Application No. 20050195832, 2005.

[116] R. Adams, *Calculus: A Complete Course*. Addison Wesley, 4th ed., 1999.

[117] "Virtex-5 Family FPGA, Overview." Xilinx Inc., available at `http://direct.xilinx.com/bvdocs/publications/ds100.pdf`.

[118] R. Norton and D. Yeager, "A Probability Model for Overflow Sufficiency in Small Hash Tables," *Communications of the ACM*, vol. 28, no. 10, 1985.

[119] T. Kurtz, "Solutions of Ordinary Differential Equations as Limits of Pure Jump Markov Processes," in *Journal of Applied Probability*, vol. 7, pp. 49–58, 1970.

[120] E. E. Johnson, "High-Speed Computation of Cyclic Redundancy Checks," Tech. Rep. NMSU-ECE-95-011, New Mexico State University, 1995.

[121] S. Goldberg, *Introduction to Difference Equations*, vol. 1. London: John Wiley and Sons, 1963.

[122] M. Spiegel, *Finite Differences and Difference Equations*, vol. 1 of *Schaum's Outline Series*. McGraw Hill, 1971.

[123] "National Library for Applied Network Research: Special Traces Archive." available at `http://pma.nlanr.net/Special/`.

[124] T. Huang, "Fast Mutual Exclusion Algorithms Using Read-Modify-Write and Atomic Read/Write Registers," in *International Conference on Parallel and Distributed Systems*, pp. 292–299, 1998.

[125] A. Agata, K. Tanaka, and N. Edagawa, "Study on the Optimum Reed-Solomon-Based FEC Codes for 40-Gb/s-based Ultralong-distance WDM Transmission," *Journal of Lightwave Technology*, vol. 20, pp. 2189–2195, December 2002.

[126] "Forward Error Correction for Submarine Systems." ITU-T Recommendation G.975, 2004.

[127] "GIGEMON Passive Optical Network Monitoring Platform." Endace Limited, overview available at `http://www.rep-tron.com/endace/GIGEMON.html`.

[128] "N2X Multi-services Test Solution." Agilent Technologies, product overview available at `http://advanced.comms.agilent.com/n2x/products/index.htm`.

[129] A. Kirsch and M. M., "The Power of One Move: Hashing Schemes for Hardware." Submitted to INFOCOM, available at `http://www.eecs.harvard.edu/~michaelm/ListByYear.html`, 2008.

[130] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An Improved Construction for Counting Bloom Filters," in *ESA*, 2008.

[131] "Micro and Opto Electronics: Cluster Review and Strategy." Scottish Enterprise Report, 2005.

[132] C. Heidarson, "Strong Growth Returns to the Market for Semiconductor IP." Gartner Dataquest, May 2005.

[133] A. A. Thompson Jr. and A. J. Strickland, *Strategic Management: Concepts and Cases*. McGraw Hill, thirteenth ed., 2003.

[134] "Internal Market Analysis Report." Aliathon Ltd, August 2005.

[135] G. Johnson and K. Scholes, *Exploring Corporate Strategy: Text and Cases*. New Jersey: Prentice Hall, first ed., 1989.

[136] M. Porter, "How Competitive Forces Shape Strategy," *Harvard Business Review*, vol. 57, pp. 86–93, 1979.

[137] R. Grant, *Contemporary Strategy Analysis: Concepts, Techniques, Applications*. Blackwell Publishing, fourth ed., 2004.

[138] B. Nalebuff and A. Brandenburger, *Co-opetition*. Profile Books, first ed., 1996.

[139] S. Dibb, L. Simkin, W. M. Pride, and O. Ferrel, *Marketing Concepts and Strategies*. Houghton Mifflin, fourth ed., 2001.

[140] P. Kjeldsen, J. Mazur, S. Malik, T. Hanson, J. Fernandez, and A. Ishiwata, "Forecast: Optical Transport Systems, Worldwide, 1999-2008." Gartner Dataquest, 2004.

[141] S. Clavenna, "IP Over Optics: Who Needs SONET?." Lightreading Web Seminar, 2005.

[142] W. Bygrave and E. Zacharakis, A, *The Portable MBA in Entrepreneurship*. Wiley, third ed., 2004.